VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

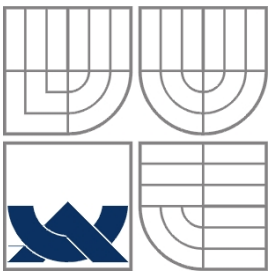
WEBOVÝ SYSTÉM PRO BUGTRACKING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

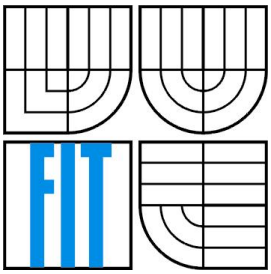
AUTOR PRÁCE
AUTHOR

Bc. JAN PREUSS

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

WEBOVÝ SYSTÉM PRO BUGTRACKING

WEB BUGTRACKING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN PREUSS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK VAŠÍČEK

BRNO 2012

Abstrakt

Práce se zabývá analýzou a návrhem nového systému pro evidenci chyb a požadavků, specifického pro prostředí FITkit a možností zachytávání a následného reportování chyb v aplikaci QDevKit. Návrh je inspirován několika hotovými nástroji, u kterých rozebírá jejich výhody a možnosti napojení na systémy pro správu a verzování zdrojových kódů. Celý navržený systém je následně implementován a důkladně popsán ve zbytku práce, včetně příkladů užitých konstrukcí a logického členění.

Abstract

Text describes some of the most common tools used for project management and bug/issue tracking. Describes advantages and possibilities of connecting them to more complex systems of administration and version control. It also describe analysis and design of new FITkit specific system, and ways to catch errors in QDevKit application. The rest of text describes whole implementation including examples of applied constructions and logical layout.

Klíčová slova

Bugtracking systémy, Trac, Bugzilla, Jira, Assembla, verzovací systémy, CVS, Subversion, Breakpad, projekt FITkit, QDevKit, QDevKit-bugs.

Keywords

Bugtracking systems, Trac, Bugzilla, Jira, Assembla, version control systems, CVS, Subversion, BreakPad, FITkit project, ODevKit, QDevKit-bugs.

Citace

Jan Preuss: Webový systém pro bugtracking, diplomová práce, Brno, FIT VUT v Brně, 2012

Webový systém pro bugtracking

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Ing. Zdeňka Vašíčka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Preuss
Datum 22.5.2012

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu Ing. Zdeňku Vašíčkovi, za odborné vedení při psaní práce a čas, který si našel pro dodatečné specifikace zadání práce. Dále bych rád poděkoval všem, kteří mi pomohli s částečnou korekturou výsledné práce.

Obsah

1 Úvod

Vývoj dnešních multiplatformních aplikací klade na jejich tvůrce poměrně vysoké nároky z pohledu ověřování funkčnosti výsledné aplikace na různých operačních systémech. To s sebou, i přes velký důraz na samotnou fázi testování, nese riziko, že se v aplikaci budou vyskytovat neobjevené chyby. Pro odladění všech podobných problémů je vhodné snažit se o jejich evidenci a v budoucích verzích jim tímto předcházet. U systémů s potenciálně velkou uživatelskou základnou je navíc žádoucí podobné reportování zpřístupnit všem uživatelům, například prostřednictvím webového rozhraní. Díky čemuž může dát libovolný uživatel snadno vědět vývojářům o chybách v konkrétních situacích.

Cílem této diplomové práce je seznámení se s existujícími systémy pro evidenci chyb a požadavků, tzv. bugtracking systémy a implementace vlastního nástroje pro školní projekt FITkit[1], který je napojen na verzovací systém Subversion[2]. Mimo to je součástí navrženého systému také implementace rozhraní pro automatické reportování informací souvisejících s pádem aplikace tak, aby bylo usnadněno následné zasílání zpráv přímo z aplikace QDevKit[3].

Pro pochopení a následný návrh požadovaného systému, je zapotřebí se detailně seznámit s již hotovými systémy a službami, které poskytují a současně diskutovat jejich výhody a nevýhody z pohledu potenciálního využití v projektu FITkit. První kapitola podrobně rozebírá několik nejběžnějších systémů na správu chyb a požadavků a srovnává je s nároky kladenými na požadované řešení. Dále popisuje dva nejběžnější centralizované verzovací systémy sloužící jako úložiště zdrojových souborů a jejich možné napojení na zmiňované bugtracking systémy pomocí jazyka PHP.

Následující kapitola se zabývá analýzou zadání diplomové práce. Popisuje tak všechny požadavky kladené na výsledný systém a jeho správné fungování. Snaží se převést stručné zadání do detailnějšího popisu konzultovaného s vedoucím práce tak, aby obsáhl všechny požadavky na tento systém.

Další obsáhlejší kapitola diplomové práce pojednává o vlastním návrhu systému a zhodnocení požadavků kladených na tento systém. Podrobně rozebírá funkční a nefunkční požadavky na systém a ukazuje využití systému z pohledu různých rolí uživatelů. Současně popisuje i databázový model tohoto systému a podrobně rozebírá propojení jednotlivých databázových entit a jejich logickou strukturu z pohledu celého systému.

---

1 *Úvod – FITkit* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <http://merlin.fit.vutbr.cz/FITkit>.

2 *Apache Subversion* [online]. Poslední úpravy 5.5.2012 [cit. 16.5.2012]. Dostupný z WWW: <http://subversion.apache.org>.

3 *QDevKit - Windows - FITkit* [online]. Poslední úpravy 20.3.2011 [cit. 16.5.2012]. Dostupný z WWW: <http://merlin.fit.vutbr.cz/FITkit/docs/navody/qdevkit.html>.

Kapitola implementace se zaměřuje na vlastní realizaci navrhovaného systému. Popisuje jednotlivé části systému členěné do celků tak, jak byly postupně implementovány. Velký důraz je zde kladen především na princip fungování jednotlivých menších částí a jejich spojování do větších celků. Zároveň kapitola ukazuje zajímavé konstrukce použité při tvorbě aplikace, které vyplynuly z návrhu systému a požadavku na přehlednost zdrojových kódů. Součástí jsou i diagramy, pro lepší znázornění popisovaných závislostí a obrázky zajímavých prvků uživatelského rozhraní, aby dokreslily celkovou představu o hotovém systému.

Diplomová práce vychází z původního semestrálního projektu, který nastínil teoretický základ celé práce a kladl si za cíl částečně analyzovat a navrhnout výsledný systém. Proto je současný návrh podrobnější oproti tomu původnímu.

2 Nástroje pro evidenci chyb a požadavků

Při práci na větších softwarových projektech, v týmu několika lidí, je nezbytné práci na projektu určitým způsobem koordinovat. Především v pozdější fázi vývoje, během období testování, je více než vhodné mít možnost spravovat podněty a nalezené chyby od testerů. Tyto podněty konzultovat a přiřazovat jednotlivým programátorům. Mnoho společností využívá softwarové nástroje pro evidenci chyb a požadavků, ať už vlastní nebo nástroje nabízené třetími stranami. Takovýchto nástrojů je velké množství, jsou napsány v různých jazycích s napojením na spoustu dalších služeb. Ale všechny pracují na podobném principu.

Základem je tzv. ticket, který reprezentuje nové chybové zprávy či požadavky. Takovýto ticket, s konkrétním problémem, může díky zmíněným nástrojům zadat většinou jakýkoliv uživatel sledovaného programu. Ticket může být přidělen přímo konkrétnímu programátorovi nebo ponechán bez přidělení. Stejně tak může být přiřazen mezi tickety pro konkrétní komponentu programu, což se liší podle zvoleného nástroje a nastavení. Jednou z důležitých vlastností každého ticketu je údaj o aktuálním stavu úlohy. V různých nástrojích jsou opět různé typy stavů, ale obecně platí několik stavů, které se v obměnách vyskytují ve všech nástrojích. Ticket tak většinou projde životním cyklem otevřeno, přiřazeno, vyřešeno a uzavřeno. Případně další rozšiřující stavy dle konkrétního nástroje. Díky tomu je přehledně vidět, kdo jaký úkol řeší a co je ještě potřeba dokončit. Součástí ticketů také často bývá nejen popis problému, ale i diskuze nad ním a možnost vkládat materiály, které se ho týkají.

Každý, dnes ve větší míře používaný nástroj, navíc umožňuje propojit tyto tickety se zdrojovými kódy na úrovni verzovacích systémů. Výsledkem je přehled vyřešených úkolů a k nim i konkrétní změny ve zdrojovém kódu aplikace, což činí práci na projektu o mnoho přehlednější. Stejně tak umí většina i určitou vzdálenou správu, aby bylo možné spravovat tickety přímo z různých, nejčastěji používaných vývojových prostředí.

Nástrojů tohoto charakteru je na světě více a bylo by nereálné popsat všechny. Proto je v následujících odstavcích vybráno několik často používaných nástrojů, které jsou dostupné úplně zdarma nebo zdarma alespoň pro nekomerční účely a je vhodné je zmínit v kontextu s řešeným problémem diplomové práce.

2.1 Trac

Nástroj Trac[4] je vyvíjený společností Edgewall Software pod modifikovanou licencí BSD. Což z něj tvoří ideální nástroj pro menší open-source projekty. Jako datové úložiště používá MySQL, PostgreSQL nebo SQLite a je kompletně napsán v jazyku Python. Nevýhodou je, že většina běžných webhostingových služeb neumožňuje spustit aplikace psané v Pythonu na svých serverech a tak na nich není možné Trac nasadit.

Trac se dá rozšířit pomocí mnoha doplňků, například pro vzdálené volání přes XML-RPC nebo napojení na různé verzovací systémy. V základní instalaci obsahuje napojení na verzovací systém Subversion, ale dokáže pracovat i s Darcs, Mercurial nebo Git.

Výhodou je snadné provázání ticketů s dokumentací ve formátu wiki, takže se uživatel může snadno odkazovat na stránky této dokumentace, popisující daný problém nebo vysvětlující výraz použitý v komentářích. Systém to tak dělá ještě přehlednější a použitelnější u velkých projektů, které jsou dostupné na webu. Možná i proto je Trac, jako nástroj pro evidenci chyb a požadavků, nasazen u projektů jako jsou Sourceforge, Beryl Project, Pidgin, Ruby on Rails a mnoho dalších. V současné době je systém uvolněn ve verzi 0.12.3, která je z června roku 2010, jak uvádí [1].

2.2 Bugzilla

Bugzilla[5] je jeden z prvních nástrojů uvolněný v roce 1998 společností Netscape Communications. V dnešní době vyvíjený společností Mozilla Foundation, která jej používá u všech svých projektů jako je Mozilla Firefox, Thunderbird a mnoho dalších. Nástroj je volně ke stažení a je distribuován pod licencí MPL (Mozilla Public License). První verze byla napsána v jazyce TCL Terry Weissmanem pro použití u projektu mozilla.org, jak se píše v [2]. Později byl nástroj přepsán do jazyka Perl, který byl tou dobou velmi populární a vznikla tak Bugzilla 2.0. Tato verze již byla uvolněna ke stažení a tak ji mohli začít využívat i běžní uživatelé.

Bugzilla má nejen webové rozhraní, jak je tomu u nástroje Trac, ale už v základě nabízí rozhraní přes e-mail, webové služby a příkazový řádek. Data ukládá do databází MySQL nebo PostgreSQL obdobně jako Trac. Stejně tak pracuje i s napojením na verzovací systémy CVS, Subversion, Perforce nebo AccuRev. Nástroj nabízí i přehledné grafy a reporty z jednotlivých fází vývoje programu a možnost pravidelného posílání reportů e-mailem.

4 *The Trac Project* [online]. Poslední úpravy 7.8.2011 [cit. 1.5.2012]. Dostupný z WWW: <http://trac.edgewall.org>.
5 *Home :: Bugzilla :: bugzilla.org* [online]. Poslední úpravy 18.4.2012 [cit. 1.5.2012]. Dostupný z WWW: <http://http://www.bugzilla.org>.

Díky tomu, že Bugzilla je jeden z prvních podobných nástrojů, je dnes hodně rozšířený. Ale i přesto se stává, že menší společnosti opouští Bugzillu a přechází například na Trac, protože je pro ně Bugzilla až příliš rozsáhlá a složitá. To je také jedna z jích hlavních nevýhod pro nasazení u menších projektů. Hodně velkých firem ji ale stále používá. Příkladem může být již zmiňovaná Mozilla nebo projekty Gnome, KDE, Open Office, Eclipse, linuxové distribuce Red Hat, Mandriva, Gentoo, ale i společnosti jako NASA či Facebook, což je jen zlomek ze seznamu [3].

V současné době je nástroj Bugzilla ve verzi 4.2.1 a jak je u společnosti Mozilla zvykem, už se podle [4] chystá uvedení verze 4.4. Nová verze opravuje několik chyb a snaží se zlepšit práci s celým systémem.

2.3 Jira

Nástroj Jira[6] vyvíjený společností Atlassian Software Systems je kompletně napsán v jazyce Java a je volně k dispozici pro nekomerční projekty. Komerční licence jsou pak zpoplatněny buď podle počtu uživatelů nebo měsíční sazbou, pokud je projekt hostován přímo na serverech společnosti Atlassian. Počátky tohoto nástroje sahají do roku 2002, kdy byla vydána verze 1.0, jak se píše na webu společnosti [5].

Uživatelské rozhraní zahrnuje webový portál, e-mail a RSS. Silnou zbraní tohoto nástroje je podpora velkého množství databázových úložišť mezi které patří DB2, Firebird, HSQLDB, MaxDB, MySQL, Oracle, PostgreSQL, SQL Server nebo třeba SybaseASA. Což je vhodné například při migraci z jednoho databázového řešení na jiné.

Jako všechny podobné nástroje obsahuje i napojení na běžné verzovací systémy. Konkrétně pak CVS, Subversion, Perforce, AccuRev a ClearCase. A také napojení na dokumentaci ve formátu wiki a další podobné rozšíření usnadňující práci v týmu a zobrazování přehledných reportů z jednotlivých fází vývoje projektu.

O kvalitě těchto nástrojů mluví vždy seznam společností, které je využívají. Ani u tohoto nástroje tomu není jinak. Podle [6] ho užívají společnosti Apache, Adobe, Cisco, Honeywell, Novell, Zend Framework, ale i jména jako CERN, NASA, Boeing nebo US Navy.

6 *JIRA- Track bugs, tasks, and projects for software development | Atlassian* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <http://www.atlassian.com/software/jira/overview>.

2.4 Assembla

Projekt Assembla[7], od stejnojmenné společnosti, se vydal odlišnou cestou než-li ostatní zmíněné nástroje. Nenabízí svůj produkt přímo ke stažení, ale směřuje k tzv. cloud computingu. Čili nabízí spravování softwarových projektů pouze on-line na svých serverech. To usnadní vývojářům menších projektů spuštění a administraci nového projektu, protože nemusí nic instalovat a pracně nastavovat.

Assembla nabízí dva základní druhy projektů. Nekomerční, který je nabízen zdarma, avšak uživatel nemůže skrýt své vlastní zdrojové soubory před ostatními uživateli. A komerční, kde už projekt vidí jen určitá skupina vývojářů, ale ten už je samozřejmě zpoplatněn.

Stejně jako předešlé nástroje umí spravovat i zdrojové kódy díky napojení na Subversion, Git nebo Mercurial. Má přehlednou práci s dokumentací ve formátu wiki a kooperaci prací na projektu usnadňuje i nástěnka nebo systém zasílání zpráv.

Sama společnost se na svých stránkách [7] chlubí 500 000 uživateli, ale jedná se spíše o menší hostované projekty, než nějaká významnější jména. Za zmínku stojí snad jen společnosti HTC nebo Sharp. Jelikož se v budoucnosti očekává rozšíření cloud computingu, je možné, že se počet uživatelů časem zvýší a přivede i větší společnosti. Zároveň se dá do budoucna předpokládat i větší konkurence v nástrojích na on-line spravování projektů.

2.5 Srovnání z pohledu využití v projektu

Všechny výše zmíněné nástroje jsou robustní a vystačí na většinu středně velkých projektů. Některé lze nasadit i u opravdu velkých projektů, ale tyto nástroje jsou většinou zbytečně obsáhlé pro spravování těch menších. Proto je vždy dobré rozmyslet se, o jak velký projekt se bude jednat a podle toho volit nástroj pro evidenci chyb a požadavků. Podstatným kritériem mohou být také používané úložiště dat nebo jazyk, ve kterém je nástroj implementován. To vše závisí na konkrétních požadavcích a sledovaných projektech.

Většina webhostingových služeb nabízí především PHP a je tedy obtížné nasadit některý z výše zmíněných systémů psaných v jiném programovacím jazyce. Stejný problém je i se školním serverem, kde jsou hostované současné stránky projektu FITkit. Tento problém by řešil projekt spravovaný on-line, například založený na serveru Assembla. Nevýhodou tohoto řešení je nemožnost zakomponovat ho do stávajících stránek projektu FITkit a především do celého konceptu navrhovaného projektu. Projekt FITkit má vlastní Subversion server a není tedy důvod požívat nástroj

7 *Assembla project workspaces to accelerate software teams, with issue tracking, GIT, SVN and collaboration | Assembla* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <http://www.assembla.com>.

jen kvůli webovému rozhraní. Proto je nutné vytvořit vlastní řešení inspirované všemi předešlými nástroji, přesně podle požadavků v zadání.

Systém musí navíc splňovat jistá specifická omezení vyplývající z požadavku automatického reportování pádů aplikace QDevKit. Tento požadavek by znamenal hledat řešení jak propojit reportování chyb z aplikace s již hotovými nástroji, případně navrhnout rozšíření implementující komunikaci mezi aplikací a těmito nástroji. V nově vyvíjeném prostředí je s tímto požadavkem možné předem počítat a tak se vyhnout zbytečným pozdějším úpravám a reimplementacím již hotového řešení.

3 Správa a verzování zdrojových kódů

Problémem rozsáhlejších projektů, vyvíjených ve větším týmu, je nutnost do určité míry řídit práci tak, aby nedocházelo k situacím, kdy budou vývojáři pracovat na stejných zdrojových kódech a tím zamezit konfliktům. Částečným řešením je rozdělit projekt na jednotlivé moduly a nechat každého vývojáře pracovat samostatně na konkrétní části. Všechny moduly musí mít předem definované rozhraní, aby je mohli ostatní vývojáři využívat a začlenit do svých modulů. V praxi je tento model většinou obtížné realizovat, protože je občas nutné zasáhnout i do společných částí projektu. Proto je vhodné využít jiné, vhodnější řešení.

V tento moment vstupují do hry systémy na správu a verzování zdrojových souborů. Jedná se o systémy, které mají za úkol zamezit jednotlivým uživatelům přepsat práci někoho jiného a současně každou změnu ukládat, aby bylo možné se kdykoliv vrátit zpět k předešlým verzím.

Samotný princip většiny verzovacích systémů je podobný. Systém eviduje jednotlivé verze všech souborů projektu v tzv. revizích. Každá revize má vlastní číslo a obsahuje všechny změny, které byly provedeny od poslední revize. Systém si tak ukládá, které řádky kódu, v jakých souborech byly změněny a kdo tyto změny provedl. Neukládá vždy všechny zdrojové kódy znovu, pouze si pamatuje změny, čímž šetří místo na disku a neplýtvá tak zbytečně zdroji, které mohou být u velkých projektů důležité. Díky zmíněnému systému verzování je snadné, kdykoliv se vrátit k jakékoliv starší verzi zdrojového kódu.

Práci v týmu dotyčné systémy usnadňují i tím, že z části umí sloučit kód od dvou vývojářů, kteří pracovali na stejném zdrojovém souboru, aniž by tím obtěžovali samotného vývojáře. Většinou dokáží sloučit jen jednoduché změny, na odlišných místech ve zdrojovém kódu, u těch složitějších, které vyvolávají konflikt, pak upozorní vývojáře, aby zasáhl. Je to hlavně proto, že systém automaticky nepozná, čí verze zdrojového kódu je správná a čí špatná. Programátor následně musí rozhodnout jaká z verzí se má použít, případně upraví zdrojový kód ručně a sloučí obě změny. Pokud ale vývojář pravidelně aktualizuje zdrojové kódy, nenastává podobný konflikt příliš často. I přes nutnost občasného zásahu programátora při slučování zdrojových kódů, je tento princip daleko jednodušší než kdyby musel všechny změny kontrolovat ručně. Automatizace je v tomto případě daleko spolehlivější než obyčejný programátor, který může způsobit například ztrátu části zdrojového kódu někoho jiného jen z nepozornosti.

Tyto systémy na správu a verzování zdrojových kódů můžeme rozdělit na dvě skupiny podle toho jestli ukládají data centrálně na jeden server nebo pracují na principu, kdy distribuují kopie celého úložiště po všech klientech. Oba tyto systémy mají své výhody i nevýhody a proto je užitečné obě možnosti podrobněji rozebrat.

3.1 Centralizované systémy

Systémy jako například CVS[8] nebo Subversion[9] využívají jedno centralizované datové úložiště na serveru a každý klient má u sebe vždy jen kopii aktuální verze. V praxi to funguje tak, že si klient aktualizuje svoji kopii projektu ze serveru a dále na ní pracuje. Má tak přístup vždy jen ke své upravené kopii a porovnávat ji může pouze s verzí, kterou si ze serveru aktualizoval. Nevýhoda je, že je systém závislý pouze na jednom centrálním místě, kde hrozí ztráta všech dat a je proto nutné pravidelně data ze serveru zálohovat. Výhodou je, že klient má pouze jednu kopii projektu a zbytek je uložen na serveru, takže se u klienta zbytečně neplýtvá prostředky, pokud pracuje například na několik projektech současně.

Pro správu velkých projektů, které mohou mít mnoho potenciálních přispěvatelů, je vhodné zpřístupnit tento repozitář prostřednictvím internetu všem, kteří na něm mohou pracovat. V této souvislosti vzniklo velké množství veřejných služeb nabízející možnost založit si volně přístupné repozitáře na jejich serverech. Mezi nejznámější a jedny z nejvyužívanějších patří například projekty Google Code[10] nebo SourceForge[11].

3.2 Distribuované systémy

Druhou skupinou jsou distribuované systémy, mezi které patří například Git[12], Bazaar[13], Mercurial[14] a další. Ty se od centralizovaných liší tím, že každý klient má u sebe celou historii projektu, takovou jako je u centralizovaných systémů uložena na serveru. Výhodou je, že má každý klient vždy kompletní historii a může ji pohodlně procházet. Navíc si tak každý u sebe udržuje zálohu celého systému, čímž se redukuje riziko ztráty dat. Z toho ale vyplývá největší nevýhoda těchto systémů, neboť projekt včetně historie může zabrat poměrně velký prostor na disku.

Další nevýhodou je, že neexistuje centrální úložiště, kde by byla dostupná aktuální verze projektu. Všichni vývojáři jsou na stejné úrovni a každá jednotlivá verze může být považována za právě aktuální verzi. S tím souvisí i nutnost distribuovat lokální změny jednoho uživatele mezi všechny ostatní uživatele. Tento princip je výhodný, pokud například na určitém modulu pracují

8 *CVS - Open Source Version Control* [online]. Poslední úpravy 3.12.2006 [cit. 1.5.2012]. Dostupný z WWW: <http://www.nongnu.org/cvs>.
9 *Apache Subversion* [online]. Poslední úpravy 5.5.2012 [cit. 16.5.2012]. Dostupný z WWW: <http://subversion.apache.org>.
10 *Google Code* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <http://code.google.com>.
11 *SourceForge - Download, Develop and Publish Free Open Source Software* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <http://sourceforge.net>.
12 *Git* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <http://git-scm.com>.
13 *Bazaar* [online]. Poslední úpravy 2010 [cit. 1.5.2012]. Dostupný z WWW: <http://bazaar.canonical.com/en>.
14 *Mercurial SCM* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <http://mercurial.selenic.com>.

pouze dva vývojáři a změny si tak vyměňují jen mezi sebou. Problémem ale je, udržet tímto stylem nějakou centralizovanou verzi, kterou budou všichni sdílet. Řešením je rozšířit skupinu uživatelů o speciálního člena, který neprovádí žádné změny, ale jsou na něj napojeni všichni ostatní uživatelé a který se chová podobně jako server u centralizovaných systémů. Tento server pak přehledně udržuje aktuální vývojovou větev projektu a nedochází tak k nesrovnalostem, čí verze projektu je ta aktuální. Existuje řada on-line služeb, které se chovají jako server v této projektové hierarchii a umožňují přehledně spravovat rozsáhlé projekty. Mezi nejznámější služby jistě patří projekty GitHub[15] nebo Gitorious[16].

Následující podkapitoly jsou zaměřeny pouze na centralizované systémy a to především na Subversion a jeho předchůdce CVS. Je to z důvodů nutnosti využít systém Subversion v navrhovaném systému, jak vyžaduje zadání diplomové práce. Ze stejného důvodu je rozebrán i princip přístupu k systému Subversion z jazyka PHP.

3.3 CVS

CVS je jeden z prvních verzovacích systémů na správu zdrojových souborů. Zkratka CVS znamená Concurent Version System, tedy systém na správu verzí. Je kompletně napsaný v jazyce C a první verze byla vydána už v roce 1990.

Princip spočívá v jednom úložišti na serveru, kterému se říká repozitář. V něm mohou být samostatné moduly pro jednotlivé projekty. Celá hierarchie projektu je na serveru uložena v běžné adresářové struktuře, kde repozitář je kořenová složka a moduly jsou její podsložky. Podsložky modulů pak přesně kopírují adresářovou strukturu konkrétního projektu.

Pokud chce vývojář pracovat na některém z modulů, připojí se k danému repozitáři a modulu. Tím se mu do předem určené složky stáhne ze serveru celá adresářová struktura projektu, se všemi aktuálními zdrojovými kódy. Takto vytvořené kopii systému se říká pracovní kopie. Vývojář na ní může pohodlně pracovat a ve chvíli, kdy je hotov, ji odeslat zpět na server, kde se vytvoří nová revize.

Každá revize jde označit štítkem, anglicky nazývaným tag. Lze tak tedy jednoduše vytvářet různá vydání celého projektu, například určovat stabilní a vývojové verze. Kdykoliv je možné vytvořit i novou vývojovou větev, zvanou branch a pracovat na ní nezávisle. Po čase je možné ji opět sloučit do jednoho projektu, podobně jak se píše například na stránkách [8].

15 *GitHub · Social Coding* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <https://github.com>.
16 *Gitorious* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <http://gitorious.org>.

Program CVS je ovladatelný pouze z příkazové řádky, ale existuje plno nadstaveb pro pohodlnou práci v různých souborových manažerech, jako třeba WinCVS. Nebo je systém CVS přímo integrován ve vývojových prostředích například Eclipse nebo NetBeans.

3.4 Subversion

Systém Subversion, zkráceně nazýván SVN, vychází ze staršího CVS a přebírá většinu jeho funkčnosti. Jde o jeden z nejrozšířenějších centralizovaných systémů na správu zdrojových kódů. I když je v poslední době částečně vytlačován distribuovaným systémem GIT, díky jeho lepší podpoře větvení projektu. Od CVS se liší především systémem uložení verzovaných souborů. Neukládá je do stromové struktury stejné jako v projektu, ale do relační databáze BerkleyDB v podobě binárních souborů. Tento způsob s sebou nese jeden problém, přímo na serveru není vidět struktura repozitáře a není lehké se k ní dostat například při poškození některého ze zdrojových souborů. Výhodou je ale větší rychlost při práci s repozitářem a především možnost uložit do verzovacího systému jakýkoliv, nejen textový, soubor. CVS také umožnila ukládat binární soubory, ale bylo třeba jim nastavovat speciální parametry. V Subversion tento problém odpadá a s binárními soubory se pracuje stejně jako s textovými.

Subversion při práci s repozitářem využívá transakce [9]. Pokud například vývojář odešle na server nová data a během nich dojde ke konfliktu, Subversion o konfliktu informuje a automaticky vrátí všechny změny zpět. Na rozdíl od CVS, které sice zaznamená konflikt, ale změny nevrátí, což ve výsledků způsobí například odeslání jen poloviny všech změněných zdrojových kódů a to není pro využití se systémem na evidenci chyb příliš vhodné. Každá nová revize v repozitáři by měla odpovídat jednomu ticketu v systému na evidenci chyb, čímž se zajistí větší přehlednost při správě projektu.

Stejně jako CVS se i Subversion ovládá pouze z příkazové řádky, ale existuje řada nástrojů pro usnadnění práce s ním. Grafické nadstavby jsou dnes dostupné pro většinu operačních systémů i vývojových prostředí jako Eclipse nebo NetBeans. Pro uživatele je tak práce s tímto systémem přehledná a pohodlná. Při integraci těchto nástrojů přímo do operačního systému, je pak možné dokonce přehledně pracovat s verzovanými složkami a soubory přímo z běžných souborových manažerů.

3.5 Napojení verzovacích systémů na PHP

Jazyk PHP zatím zcela nepodporuje žádný z verzovacích systémů, ale existuje několik řešení jak s nimi pracovat. Následující text se bude zabývat možnostmi napojení na PHP pouze u systému Subversion, protože je využíván v projektu FITkit. U ostatních systémů je ale situace obdobná a žádná ze současných implementací není zcela ideální a zahrnuta v základní instalaci PHP.

Možností jak přistupovat k systému Subverison ze skriptovacího jazyka PHP je hned několik. Ale ani v tomto případě není žádná z nich integrovaná přímo v instalaci. Je tedy nutné vždy doinstalovat potřebná rozšíření, mít možnost spouštět externí programy nebo si vytvořit vlastní interface pracující se Subversion přes jedno z jeho běžných rozhraní.

Jednou z možností je přistupovat k datům v Subversion přes rozhraní WebDAV[17], které systém umožňuje. To ale vyžaduje zásah do webového serveru kvůli nutnosti nastavit WebDAV server volající samotný Subversion. Což znamená mít dostatečná práva a nebo o tuto možnost požádat administrátora serveru. Následně je potřeba napsat si vlastní způsob jak přistupovat z PHP k WebDAV službě a to včetně práce s rozšířením WebDAV DeltaV. DeltaV je nadstavba klasického WebDAV serveru o možnost spravovat verze jednotlivých souborů. Je poměrně rozsáhlá a implementovat její plnou funkčnost v PHP by bylo dost náročné.

Další možností je doplnit do PHP například modul *svn* [10], který umí se Subversion pracovat. Aby to bylo možné na serveru provést, je opět zapotřebí administrátorského účtu. Nevýhodou je, že PHP zatím nenabízí plnou podporu pro žádný podobný modul. Hrozí tak, že některé funkce mohou změnit své názvy, přijímané parametry nebo být úplně odstraněny v dalších vývojových verzích. Vývojové verze sebou nesou vždy i určité množství chyb, které se v průběhu teprve odstraňují, což není žádoucí pro nasazení v běžných projektech. Řešení je to ovšem elegantnější, protože programátor už pak pouze využívá třídu pracující se Subversion a nemusí se starat o komunikaci se systémem samotným.

Jelikož PHP podporuje spouštění příkazů na serveru, je možné k systému Subverison přistupovat také přímo voláním příkazu svn například pomocí PHP funkce *system()*. Díky tomu lze využít všechny výhody nabízené systémem Subversion, bez nutnosti je implementovat čistě v PHP. Práce s verzovanými soubory je tak přenechána samotnému Subversion a v PHP je řešeno pouze jeho volání a zpracování výstupů, které systém vrací. Nevýhodou je, že většina webhostingů nedovoluje z PHP spouštět příkazy vůbec nebo jen určité jednoduché příkazy. Je to dáno bezpečnostní politikou jednotlivých serverů, aby uživatel nemohl spustit libovolný program a tím ohrozit celý server.

17 *WebDAV Resources* [online]. Poslední úpravy 21.4.2010 [cit. 1.5.2012]. Dostupný z WWW: <http://www.webdav.org>.

Ani jedno z popisovaných řešení není ideální a závisí na možnostech serveru a jeho administrátorovi. Proto je vhodné vždy volit takové řešení, které je pro danou situaci nejideálnější. V rámci zadání diplomové práce je specifikován i možný přístup k Subversion a to pouze voláním programu z příkazové řádky. Je to dáno především politikou školního serveru, která dovolí spouštět ze skriptů jen příkaz svn. Což je i důvod, proč není možné na školím serveru provozovat jeden z výše zmíněných nástrojů.

4 Reportování chyb a pádů aplikací

Při vývoji rozsáhlých multiplatformních aplikací s velkou uživatelskou základnou, je téměř nemožné testovat aplikaci na všech možných operačních systémech a jejich konfiguracích. Proto je občas možné při používání aplikace narazit na zatím neobjevenou chybu, která může v krajním případě vést až k pádu celé aplikace. V takovém případě je vhodné pád zachytit a dát uživateli vědět, aby nebyl zbytečně a nemile překvapen tím, že se aplikace sama ukončila a on nevěděl proč. Zároveň je vhodné o této chybě a jejich okolnostech informovat i vývojáře dané aplikace, aby mohl chybu včas vyřešit. Protože je každý operační systém specifický svou konfigurací či ostatními nainstalovanými aplikacemi, je dobré spolu s informací o chybě odesílat i základní informace o verzi systému či používaných knihovnách. Vývojáři to tak dá lepší přehled o okolnostech, které k pádu aplikace vedly.

Nejjednodušší způsob jak dosáhnout požadovaného efektu je zachytávat předem definované výjimky za běhu programu a odesílat při nich požadované informace. Tímto způsobem však není možné zachytit kritické pády, při různých systémových voláních nebo pády zaviněné externími zdroji. Zároveň je možné zachytit pouze předem definované chyby, čily takové chyby, které vývojář při psaní předpokládal. A pokud je předpokládal, většinou se je snažil i ošetřit a řádně otestovat. Většina pádů aplikace je ale způsobena chybou, se kterou vývojář předem nepočítal a tak se tento způsob v podstatě míjí účinkem i když je jistě užitečný a částečný přínos má.

Dalším způsobem je navázat akce na různá systémová volání a tímto způsobem čekat, jestli se některá z nich provedou. Pomocí této techniky lze zachytit i fatální pád aplikace a ihned na něj zareagovat. Vývojář tak má plnou kontrolu nad během a případným pádem aplikace i když provede fatální chybu. Nevýhodou pro multiplatformní aplikace je, že každý operační systém a každá jeho verze má odlišné volání těchto akcí. Je tedy nutné počítat s různými operačními systémy a pro každý zvlášť psát zvláštní kus kódu aplikace. Následně také vše důkladně testovat na různých platformách. To ale stojí vývojáře příliš sil a není to moc efektivní, pokud se nejedné o velký systém s potenciálním nasazením u tisíců uživatelů na rozličných platformách.

Nabízí se tak poslední varianta, jak zachytávat chyby a výjimky způsobené chystanou aplikací. Využít některé z již vyvinutých nástrojů, které jsou dobře testované a odladěné pro nejrůznější operační systémy. Podobných nástrojů je velké množství a liší se jak způsobem zachytávání pádů aplikací, tak počtem informací, které jsou při pádu schopné zjistit, případně programovacím jazykem pro který jsou napsány. Většinou jsou ale navrženy pro konkrétní účely a aplikace a je třeba si je pro své potřeby uzpůsobit. Mezi nejpoužívanější patří například Windows Error Reporting[18],

18 *WER Functions (Windows)* [online]. Poslední úpravy 7.2.2012 [cit. 1.5.2012]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/bb513635.aspx>.

Bug Buddy[19] pro projekt GNOME nebo knihovna CrashRpt[20] pro zachytávání chyb v prostředí Microsoft Visual C++. Bohužel, jen málo z nich je uvolněno pod svobodnou licencí a není tedy možné je pro potřeby tohoto projektu využít. V tomto směru vyniká nástroj Breakpad[21], kterému se věnuje následující podkapitola této práce.

4.1 Breakpad

Aplikace Breakpad je vyvíjena společností Google a je šířena jako open-source pod BSD licencí, takže ji mohou do svých projektů začlenit i ostatní vývojáři. Kromě možnosti tento nástroj volně využívat, je jeho výhodou i zastoupení pro většinu běžných platforem, což je užitečné při vývoji univerzální aplikace v některém z multiplatformních vývojových prostředích jako je třeba Qt. Celý nástroj je napsán v jazyce C++ a je tak snadno použitelný. Výhodou je, že Breakpad se nesnaží vytvořit jedno řešení pro všechny platformy a tím se uchýlit k jistým kompromisům, ale na každé samostatné platformě se chová specificky. Výsledkem je navenek sjednocené API pro zachytávání výjimek, které je vnitřně uzpůsobené konkrétní platformě.

Princip fungování je založený na automatickém generování tzv. dump souborů s informacemi o pádu, včetně adres paměti, na kterých program skončil a názvů funkcí volaných v době pádu. Tento dump má jednotný formát na všech systémech a je tedy pro vývojáře ideální, protože se nemusí starat o několik nezávislých formátů.

Nástroj tvoří tři základní části, knihovna v klientské aplikaci, program na generování tabulky symbolů a procesor na zpracování dump souborů. Jednotlivé části mají pevně definované vstupy a výstupy tak, aby spolu dokázaly spolupracovat a tvořit tak jeden velký celek usnadňující práci.

Knihovna v klientské aplikaci se stará o zachytávání pádů aplikace na konkrétních platformách a tyto informace zapisuje do souboru ve speciálním formátu tzv. minidump souborů. Nástroj už dál explicitně neřeší jakým způsobem se soubory dostanou od uživatele k vývojáři, to je na vývojáři samotném. Formát minidump souboru je navržen společností Microsoft a je podobný core dump souboru známému z linuxových systémů. Zvolen byl především proto, že je menší než core dump soubory a snazší pro použití na platformě Windows. Zároveň je také lépe dokumentovaný než zmiňované core dump soubory na Linuxu [11].

19 *Bug Buddy in Launchpad* [online]. Poslední úpravy 29.9.2010 [cit. 1.5.2012]. Dostupný z WWW: <https://launchpad.net/bug-buddy>.

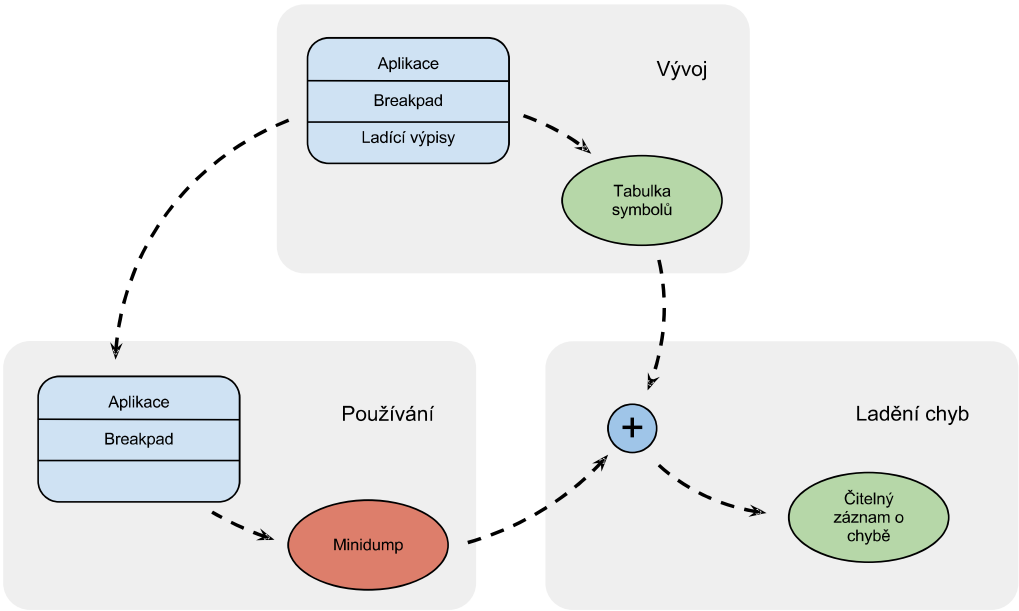
20 *crashrpt - A crash reporting system for Windows applications - Google Project Hosting* [online]. Poslední úpravy 22.10.2011 [cit. 1.5.2012]. Dostupný z WWW: <http://code.google.com/p/crashrpt>.

21 *google-breakpad - Crash reporting - Google Project Hosting* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <http://code.google.com/p/google-breakpad>.

Aby bylo možné podle adres v paměti určit, ve které funkci se program před pádem nacházel, je nutné pro každý program vytvořit tzv. tabulku symbolů. K jejímu generování slouží další část nástroje Breakpad, program *dump_syms*. Tento program generuje pro adresy v paměti jména funkcí tak, aby bylo při rekonstrukci možné zobrazit i volání jednotlivých funkcí a čísla řádků ve zdrojových souborech.

Tabulku symbolů vytváří vývojář při každém uvolnění nové verze, ale nechává si ji pro sebe. Slouží pouze k zpřehlednění zasílaného výstupu a není třeba ji distribuovat všem uživatelům. K interpretaci minidump souboru v kombinaci s nachystanou tabulkou symbolů slouží poslední část nástroje Breakpad, program *minidump_stackwalk*. Ten se stará o přehledné a čitelné zobrazení informací o pádu vývojáři. Tuto část má k dispozici opět jen vývojář a není třeba ji distribuovat uživateli.

Postup při vydání nové verze aplikace, její distribuce uživateli a zpracování minidump souboru je vidět na obrázku 1. Vývojář nejprve přeloží aplikaci s povolením ladících výpisů a z aplikace nechá vygenerovat tabulku symbolů. Poté nabídne uživatelům aplikaci již bez povolených ladících výpisů. Když aplikace na uživatelově systému provede nějakou neplatnou operaci a ukončí se, je vygenerován minidump a odeslán vývojáři. Ten si ho nechá zobrazit společně s tabulkou symbolů, získanou z ladících informací a přesně ví, ve které funkci a na kterém řádku došlo k chybě.



Obrázek 1: Zpracování pádů aplikace

5 Projekt FITkit

Na Fakultě informačních technologií Vysokého učení technického v Brně vznikl projekt FITkit, který si klade za cíl přiblížit studentům technologie hardware a software výpočetních systémů. Přitom je kladen důraz především na praktickou stránku věci než jen strohou teorii. Základem projektu je hardwarové zařízení obsahující výkonný mikrokontrolér s nízkým příkonem, hradlové pole FPGA a možnost rozšíření zařízení o další periferie jak se píše v [12]. Výhodou hradlového poje je jeho snadné programování v jazyce VHDL, který je pro studenty přívětivější než například programování přímo v assembleru. Díky tomu je možné programy nejprve odsimulovat a až po odzkoušení nahrát finální verzi do samotného hardwarového zařízení.

Do celého projektu se mohou jednoduše zapojit i jednotliví studenti a přispět tak k jeho vývoji. Celý projekt se neskládá pouze ze zmiňovaného hardwarového zařízení, ale i s programů a nástrojů, které usnadňují práci se zařízením. Jedná se především o komplexní aplikace QDevKit.

5.1 QDevKit

Aplikace QDevKit je multiplatformní grafické uživatelské rozhraní pro snadnou komunikaci a práci s hardwarovým zařízením FITkitu. Díky ní je možné jednoduše do zařízení nahrávat vytvořené programy. Pro vlastní běh využívá další část projektu, knihovnu libkitclient, která slouží k odstínění práce s připojeným zařízením a nabízí rozhraní pro komunikaci s ním. Tím vytváří univerzální nástroj, jež je použitelný nezávisle na operačním systému uživatele.

QDevKit je kompletně napsaný v jazyce C++ s využitím knihovny Qt pro návrh grafického uživatelského rozhraní. Navíc obsahuje knihovnu pro práci s jazykem Python, takže je možné psát doplňky i v tomto skriptovacím jazyce. Celý projekt je open-source, takže jej může kdokoliv nezávisle rozšiřovat a tím neustále zlepšovat.

6 Analýza systému

Cílem projektu je vytvořit systém na evidenci chyb a požadavků spojených se softwarem QDevKit pro FITkit. Systém by měl být založen na technologii PHP, využívat databázový sever MySQL a být integrovatelný do již dostupných stránek projektu FITkit. Správa chyb a požadavků by měla být úzce svázána s úložištěm zdrojových souborů Subversion, aby bylo možné evidovat jednotlivé úpravy. Napojení na Subversion musí být implementováno přes volání příkazu svn ze skriptů PHP.

Uživatelé systému budou rozděleni do tří skupin s různými oprávněními pro přístup k systému. Běžným uživatelům systém umožní reportovat chyby, případně vkládat nápady na nová rozšíření a komentovat je. Zároveň jim nabídne možnost procházet i celé úložiště zdrojových souborů. Druhou skupinou budou udržovatelé balíčků se stejnými právy jako běžní uživatelé, rozšířenými o potvrzování uzavření verzí. Nejvyšší oprávnění připadnou na skupinu správců. Ti zdědí stejná oprávnění jako uživatelé s tím rozdílem, že navíc získají možnost spravovat jednotlivé příspěvky, přiřazovat jim vlastníky a uzavírat potvrzené verze systému. Běžnými uživateli budou všichni studenti fakulty a její zaměstnanci. Udržovatelů balíčků bude jen několik a uživatel s administrátorskými právy by měl být v celém systému pouze jeden.

Systém musí obsahovat správu verzí software zahrnující výše zmíněné příspěvky, ať týkající se chyb, tak nově navrhovaných rozšíření. Každá uzavřená verze obsáhne všechny v ní vyřešené chyby i přidaná rozšíření a tato informace by měla být dostupná i ve formě například XML pro využití v samotné aplikaci. Uzavření jednotlivých verzí bude probíhat v několika krocích. Nejprve správce nastaví dané verzi příznak, že je uzavřena a udržovatelé balíčků jsou na to upozornění e-mailem. Následně každý z nich po přeložení software pro svojí platformu, potvrdí tento příznak v systému. Teprve až všichni udržovatelé balíčků potvrdí příznak, je možné verzi uzavřít a vydat informace o nové verzi.

Současně je třeba rozšířit stávající aplikaci QDevKit o nástroj na zachytávání jejích pádů a chyb a automatické reportování těchto problémů přímo do navrhovaného systému. Odesíláno by mělo být vše včetně informací o okolnostech pádu, platformy kde se chyba vyskytla a uživateli, který tuto chybu zaznamenal. Aby tak bylo možné pružněji reagovat na chyby nalezené uživateli při používání programu QDevKit.

Důraz bude kladen na přehlednost systému, jeho snadnou obsluhu s ohledem na bezpečnost a možnosti dalšího rozšíření v budoucnu. Nejen z tohoto důvodu je požadován srozumitelný a čistý kód, tak aby s ním bylo možné dále pracovat. Jelikož jsou stávající stránky projektu FITkit ve více jazycích, je nutné na tuto skutečnost připravit i výsledný systém. Aktuálně je třeba počítat s českou a anglickou jazykovou verzí.

7 Návrh systému

Následující podkapitoly popisují jednotlivé fáze návrhu systému, který vychází z výše zmíněné prvotní analýzy. Tento návrh se detailně zabývá popisem systému a jeho funkcemi. Zároveň podrobně rozebírá veškeré požadavky kladené na systém z pohledu návrhu aplikace. Popisuje případy užití a práce se systémem stejně tak jako samotný návrh datové struktury celého systému.

Slovník pojmů

Při návrhu požadované aplikace byla použita terminologie popsaná níže, aby se tak usnadnila orientace v následujícím textu a předešlo se případným nedorozuměním při různé interpretaci stejných termínů.

Ticket - Položka v systému reprezentující buď chybu nebo nové rozšíření systému
Revize - Soubor změn ve zdrojových kódech od poslední synchronizace
Verze - Soubor několika vyřešených ticketů uzavřených v jedné skupině
Návštěvník - Nepřihlášený uživatel zobrazující stránky projektu
Uživatel - Běžný uživatel (nejčastěji student) s možností vytvářet tickety a přispívat k nim
Udržovatelé balíčků - Uživatel s možností potvrzovat uzavření verzí
Správce - Uživatel s možností spravovat tickety a verze systému
Závažnost - Vlastnost přidávaného ticketu určující jeho důležitost
Platforma - Operační systém, na kterém uživatel objevil závadu
Dump - Soubor s podrobnými informacemi o pádu aplikace

7.1 Funkční požadavky

Na základě analýzy požadavků na danou aplikaci byly stanoveny funkční požadavky sepsané níže. Tyto požadavky jsou členěny dle rolí uživatele v systému.

Návštěvník může provádět následující operace:

1. Procházet jednotlivé revize úložiště zdrojových souborů
2. Vypisovat obsah souborů v úložišti zdrojových souborů
3. Zobrazovat informace o všech verzích

Uživatel může provádět stejné operace jako návštěvník rozšířené navíc o následující operace:

1. Vytvářet tickety
2. Procházet tickety jiných uživatelů
3. Přispívat ke všem ticketům
4. Vytvoři ticket po pádu aplikace QDevKit odesláním informací o pádu

Udržovatel balíčků má stejné možnosti jako uživatel, rozšířené o následující činnosti:

1. Potvrdit vytvoření balíčku a uzavření verze
2. Editovat tickety
3. Přiřazovat tickety udržovatelům
4. Přiřazovat tickety k verzím

Správce je speciální typ udržovatele balíčků s dalšími rozšířenými možnostmi:

1. Spravovat uživatelské účty
2. Navrhnout uzavření verze
3. Uzavřít připravenou verzi
4. Přidávat nové verze

Speciální rolí je samotný systém který dělá sám následující činnosti:

1. Automatické vytváření revizí podle systému Subversion

7.2 Nefunkční požadavky

Na základě analýzy požadavků na danou aplikaci byly stanoveny i tyto nefunkční požadavky, které zahrnují předpoklady pro základní běh systému. Předpokládaná zátěž systému je pouze orientační. Systém by neměl být striktně vázán na určitou zátěž, ale navržen tak, aby vydržel i větší zatížení.

1) Databázový systém MySQL
2) Webový server (Apache, IIS…)
3) Skriptovací jazyk PHP 5.1.6
4) Nainstalovaný server Subversion včetně klienta pro příkazový řádek
5) Webový prohlížeč podporující webové standardy (nejlépe Mozilla Firefox)
6) Podpora cookies v prohlížeči
7) Podpora JavaScript v prohlížeči

8) Předpokládaná zátěž systému je do 100 návštěvníků za den
9) Předpokládaný počet návštěvníků připojených v jeden moment je do 20
10) Předpokládaný počet aktivních udržovatelů balíčků je do 20
11) Předpokládaný počet administrátorů je do 3

7.3 Případy užití

Příloha A zobrazuje diagram případu užití rozdělený na jednotlivé role uživatelů v systému tak, aby byla zřetelně vidět jejich hierarchie. Diagram se nesnaží zobrazit všechny detailní činnosti, ale obecný princip jednotlivých činností. Některé činnosti jsou proto seskupeny do jedné obecnější činnosti pro snazší orientaci a představu jak celý systém funguje.

Následující případy užití ukazují dvě, z pohledu uživatele, zajímavé situace. Vytváření nových ticketů a uzavírání vývojové verze včetně potvrzení hotových balíčků.

7.3.1 Vytvoření nového ticketu

Vytváření nových ticketů je možné pouze pokud je uživatel přihlášen a je znázorněno v příloze B. Uživatel zvolí možnost vytvořit nový ticket a následně mu systém nabídne formulář na jeho založení. Uživatel vyplní požadované údaje a má možnost k ticketu připojit soubor nebo vlastní komentář. Musí také vybrat operační systém, ke kterému se ticket váže. Položka operační systém je ve většině případů předvyplněna, ale uživatel může tuto hodnotu určit ručně, pokud by automatické rozpoznávání operačního systému bylo nepřesné. Pokud uživatel nevyplní některý z povinných údajů, je o tom informován, jak je vidět na alternativním toku v příloze C. Až když uživatel vyplní vše správně, vytvoří systém v databázi nový ticket a nastaví mu status na „nový“.

7.3.2 Uzavření verze

Uzavření verze je detailně popsáno v případu užití v příloze D. Podílí se na něm správce a všichni udržovatelé balíčků k verzi dříve přiřazení. Správce označí verzi za připravenou k uzavření, čímž se odešle e-mailem zpráva všem udržovatelům balíčků. Poté správce čeká na potvrzení od všech udržovatelů balíčků. Ti musí přeložit balíček pro svou platformu a potvrdit tuto skutečnost prostřednictvím systému. Teprve až tak učiní všichni, správce může verzi uzavřít. Systém pak bude takto nově vytvořenou verzi nabízet společně se všemi předešlými v informačním kanálu ve formátu XML.

7.4 Návrh řešení

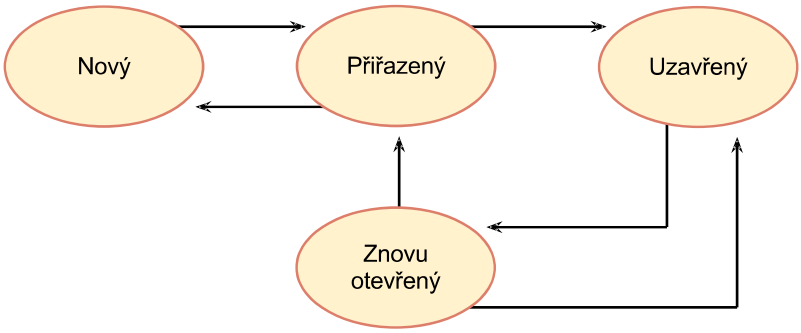
Projekt je rozdělen do tří pomyslných částí, které jsou co do realizace a potřebných implementačních nástrojů dosti odlišné, ale navzájem spolu spolupracují. Jedná se o dvě části dohromady tvořící webové rozhraní, tedy celý systém pro reportování chyb a požadavků a zobrazení obsahu samotného repozitáře projektu FITkit. Třetí část je specifická především nutností začlenit ji do již stávající aplikace QDevKit a napsat ji tedy v jazyce C++, ale zároveň se systémem komunikovat podobně, jako skrze jeho webové rozhraní.

7.4.1 Reportování chyb

Hotový systém bude fungovat pouze jako modul stávajících stránek projektu FITkit, proto není nutné tvořit celý samonosný systém, ale naopak je třeba jej začlenit do stávajících stránek. Nevýhodou je striktnější specifikace vytvořeného prostředí. Je třeba dodržet použitou strukturu stránek i odkazů mezi nimi. Základní struktura je složena ze skriptu *index.php* a jednotlivých šablon vypisujících požadované informace. Skript *index.php* slouží jako tzv. kontrolér, tedy skript starající se o jednotlivé akce systému a pracující s databázovým modelem. Z něj jsou pak načítané jednotlivé šablony starající se o vypsání požadované stránky s daty.

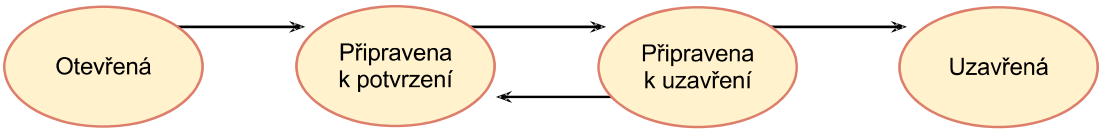
Autentizace uživatelů v systému se řeší už na úrovni stránek projektu FITkit a není třeba ji tedy explicitně řešit v navrhovaném systému. Skriptu *index.php* je vždy předána informace, zda je uživatel přihlášen či nikoliv a také podrobnější informace o tomto uživateli. Z tohoto důvodu není zapotřebí, aby si systém uchovávat v databázi záznamy pro všechny přihlášené uživatele, ale stačí si pamatovat ty uživatele, kteří mají rozšířená oprávnění. Ušetří se tak čas a samozřejmě i prostor vyhrazený pro databázi. Současně se tak předejde zbytečné manipulaci se stovkami uživatelských účtů vždy, když nastoupí noví studenti nebo ti současní studium ukončí.

Systém bude pracovat s jednotlivými tickety obdobně jako většina podobných systémů. Nový ticket bude moci přidávat libovolný přihlášený uživatel. Stejně tak i psát komentáře k ticketům. Pracovat s ticketem, jako měnit vlastníka nebo stav, budou moct pouze uživatelé s rozšířenými právy. Životní cyklus ticketu bude popsán čtyřmi stavy, jak je vidět na obrázku 2. Zobrazení ticketů bude možné filtrovat podle různých kritérií, ale prvotní zobrazení bude ukazovat chronologicky seřazené a zvýrazněné tickety, seskupené podle závažnosti a stavu. Zároveň bude nabízet zobrazení aktuální i všech ostatních verzí a vypisovat všechny změny, které se v nich udály. Provázání ticketů a revizí v Subversion bude probíhat obdobně jako například u systému Trac, tedy pouze textovým odkazem na danou revizi. Bude se tak jednoduše možné odkazovat z komentářů v revizích zpět na tickety a nebude nutné při načítání nových revizí vytvářet v databázi pevné vazby.



Obrázek 2: Životní cyklus ticketu

Životní cyklus jednotlivých verzí bude popsán čtyřmi stavy zobrazenými na obrázku 3. Podrobný popis postupu při změně stavů je názorně popsán v předešlém případu užití. Informace o jednotlivých uzavřených verzích systému tak, jak budou postupně uvolňovány, bude dostupná ve formátu XML. Nikoliv přes statický soubor generovaný při uzavření nové verze systému, ale přístupný z určité adresy, aby byla poskytovaná informace vždy aktuální. Docílí se tím nabídnutí přesných informací o nové verzi a seznamu změn ve starších verzích. Takto vygenerovaná XML informace bude snadno použitelná v aplikaci, aby bylo možné uživatele informovat o nově uvolněné verzi používaného programu.



Obrázek 3: Životní cyklus verze

7.4.2 Procházení repozitáře

Pro práci se systémem Subversion by bylo nejvhodnější použít samostatnou třídu tvořící rozhraní mezi programem svn spouštěným z příkazové řádky a PHP. V rámci této třídy implementovat veškerou funkcionalitu a v PHP jen využít její rozhraní pro pohodlnější práci se Subversion. Nebude tak třeba zbytečně duplikovat celou databázi revizí a informací o nich, ale systém bude tyto informace načítat ze samotného Subversion. Protože ale systém Subversion nedovoluje modifikovat již jednou provedené změny, není nutné se na stejná data ptát pokaždé znovu. Tento postup by byl příliš pomalý a znamenal by zbytečné prodlevy při načítání. Proto je vhodné použít systém ukládání výsledků volání příkazů Subversion do určité lokální vyrovnávací paměti, například ve formě souborů

na disku. Ale pouze tehdy, až bude tento příkaz požadován. Nebudou se tak zbytečně na disk ukládat všechny informace ze Subversion, ale pouze ty, které budou potřeba tak, aby při příštím vyžádání byly znovu rychle dostupné. Zamezí se tím zdlouhavému načítání již jednou zobrazených dat a přitom se minimalizuje problém se zbytečným duplikováním všech dat uložených v systému Subversion. V samotné databázi bude uloženo vždy pouze číslo a popis jednotlivých revizí, aby se s nimi dalo dále jednoduše pracovat a procházet je.

7.4.3 Aplikace QDevKit-bugs

Na první pohled samostatnou částí řešení je realizace způsobu zachytávání chyb a pádů aplikace QDevKit a reportování těchto stavů do navrhovaného systému. Pro samotné zachycení všech neplatných stavů aplikace, je vhodné do současné aplikace QDevKit doplnit část kódu, která se o to bude starat. Jako nejvýhodnější se pro tento účel jeví využít nástroj Breakpad popisovaný v předchozích částech práce. Jeho nasazení do již hotového projektu je poměrně jednoduché a přitom splňuje i všechny další podmínky vyplývající ze současné implementace aplikace QDevKit. Je psaný v jazyce C++ a je multiplatformní, takže výborně zapadá do celé koncepce aplikace QDevKit.

Při pádu se nástroj Breakpad sám postará o zachycení všech chyb a vygenerování dump souboru se všemi potřebnými informacemi o pádu aplikace a zavolá jednoduchou funkci, která se postará o spuštění nového programu pro odeslání těchto informací do navrhovaného systému. Aplikace by měla být samostatná především proto, že při zachycení pádu se může aplikace nacházet ve stavu, kdy například přistupuje do nepovolené části paměti a proto by se už ani odesílání chyb nemuselo provést. Tudíž je vhodné provádět v rámci funkce ošetřující chyby jen to nejnutnější, tedy zavolat externí nástroj pro odeslání dat na server.

Aplikace QDevKit-bugs se bude skládat pouze z jednoho dialogu informujícího uživatele o tom, že při vykonávání nastala chyba. Pomocí tohoto dialogu bude možné, po vyplnění přihlašovacích údajů a volitelně i komentáře, odeslat vygenerovaný dump soubor na server. Přičemž se aplikace bude chovat podobně jako webová část systému a bude tedy odesílat soubor i všechny ostatní informace prostřednictvím složeného požadavku POST na server. Obdobně jako by to udělal uživatel ručně při odesílání formuláře pomocí webového rozhraní. Díky tomu není třeba na serverové straně řešit několik různých přístupů, jak vytvářet nové ticekty.

Aplikace bude muset sama řešit HTTP autentizaci, kterou běžně za uživatele obstarají internetové prohlížeče, ale k tomu je možné snadno využít možnosti nabízené použitým vývojovým prostředím Qt. Jeho součástí je několik tříd řešících síťovou komunikaci s webovými servery přes protokol HTTP.

7.5 Návrh databáze

Ačkoliv je navržený databázový model postaven na technologii MySQL, jak požaduje zadání systému, lze ho po menších úpravách použít i na jiných platformách. ER diagram, zobrazující rozložení a propojení jednotlivých databázových tabulek, je připojen v příloze E. Všechny tabulky jsou typu InnoDB, kvůli jednodušší práci s vazbami mezi tabulkami a možností využívat transakcí při zpracovávání dotazů nad databází. Znaková sada tabulek je jednotně navržena v kódování UTF-8, aby bylo možné ukládat do databáze nejen všechny české znaky, ale v případě potřeby i znaky jiných národních abeced, jako je například slovenština. Primární porovnávání nad tabulkami je nastaveno na *czech_ci*, tedy české řazení a porovnávání, bez ohledu na velikost znaků. Návrh je pomyslně rozdělen na dvě části, které na sebe nejsou napojeny. O toto provázání se stará samotná aplikace a uživatelé systému.

Jednodušší, samostatnou částí návrhu, je ukládání jednotlivých revizí do databáze, aby s nimi bylo možné v systému snadněji pracovat a především je hierarchicky řadit podle času vzniku. Tabulka revizi obsahuje název revize, její číslo a datum, kdy byla vytvořena.

Daleko složitější je struktura druhé části systému pro evidování chyb a požadavků. Základním prvkem celého systému jsou tickety a stejně tak je tomu i u návrhu databáze. Aby ale bylo možné pamatovat si všechny provedené změny a nejen aktuální stav ticketů, je zde zavedena tabulka s historií jednotlivých změn. Tabulka ticketů tak sama o sobě obsahuje jen minimum informací a ostatní informace jsou uloženy právě v tabulce *history*.

Myšlenka spočívá v tom, že pokud se ticket jakkoliv změní, vytvoří se nový záznam v tabulce *history* a naváží se na něj všechny změny. Princip je navržen tak, že se vytvoří nové záznamy v tabulkách všech změněných informací a do tabulky *history* se uloží jejich identifikátor. Pokud je změněna pouze jedna informace, například status, ostatní vazby zůstanou prázdné. Naopak přiloží-li uživatel k ticketu například soubor, může vytvořit i komentář nebo změnit stav. Všechny tyto změny se pak uloží do databáze jako jeden záznam v historii. Dá se tak dohledat, jak celá historie ticketu, tak například pouze historie přidaných komentářů. Tento princip napojení událostí na historii je možné rozdělit na dva způsoby. Zaprvé na tabulky obsahující přednastavené hodnoty jako jsou statusy nebo platformy. A zadruhé tabulky, do kterých je vždy nejprve přidán nový záznam a teprve pak jsou napojeny na historii. Je tomu tak například u příloh nebo již zmíněných komentářů.

Specifičtějším případem je určení verze do které ticket patří. Princip vazby na historii ticketu je obdobný jako u statusů, ale samotná tabulka verzí je o něco obsáhlejší. Především má sama svůj vlastní status udávající, zda je verze uzavřená či nikoliv. A také vazbu na aktéry, kteří se na uzavírání podílejí. Aby bylo možné určit, zda už všichni aktéři nastavili příznak k uzavření, je třeba si ještě podobný příznak jako u verzí, udržovat i u těchto aktérů. K uzavření verze tedy může dojít pouze

tehdy, mají-li všichni její aktéři nastaven status na uzavření. Každý aktér může při uzavření k dané verzi doplnit komentář, tudíž na něj bylo třeba myslet i v návrhu databázové struktury.

Každý ticket má vždy uvedeného autora a z historie dostupného i vlastníka jemuž je přiřazen. Tyto dva typy jsou na úrovni databáze odlišné, i přes to, že v reálu jsou oba aktéři ze stejné skupiny. Je to proto, aby nebylo nutné ukládat si všechny přihlášené uživatele, když se o to starají stránky FITkitu, do kterých je systém integrován. Proto je položka autor jen textová hodnota obsahující unikátní login v rámci celé školní sítě. Obdobně je tomu i s vlastníkem, ale s tím rozdílem, že seznam vlastníků je navíc uložen v tabulce *users* a je předdefinovaný správcem systému. Tabulka *users* tak eviduje jen dvě nejvyšší role v systému. Běžný uživatel není v tabulce *users* vůbec uložen na rozdíl od udržovatelů balíčků a správce systému. Zároveň se v tabulce *users* udržují i podrobnější informace o jednotlivých uživatelích, včetně e-mailové adresy, na kterou systém automaticky zasílá reporty.

Všechny doplňující informace ticketů, které je třeba udržovat ve formě číselníků, jako například typy ticketů, platformy nebo třeba závažnosti, mají v databázi samostatnou tabulku. Je to především proto, aby bylo do budoucna snazší rozšířit všechny tyto číselníky o jazykovou mutaci v případě pozdější lokalizace do více jazyků. Každá tato tabulka obsahuje svůj unikátní identifikátor v podobě ID, název hodnoty v nespecifikovaném jazyce a svou prioritu. Priorita slouží jak k řazení důležitosti jednotlivých položek, tak i k jejich identifikaci v systému. Dává tak položkám v databázi jistý sémantický význam a usnadňuje práci s nimi.

8 Implementace

Z pohledu implementace je možné rozdělit celý projekt na dvě hlavní části. První je systém na hlášení chyb a požadavků s webovým rozhraním, který je kompletně napsaný v jazyce PHP s využitím databázového systému MySQL. Druhou částí je zachytávání pádů aplikace QDevKit a reportování chyb včetně dump souborů. Tato část je kompletně napsána v jazyce C++ s využitím frameworku Qt tak, aby aby poskytovala multiplatformní rozhraní.

Při práci s PHP bylo třeba počítat s jistými omezeními. Jednalo se především o verzi PHP 5.1, která byla nutnou podmínkou z důvodů nasazení na školní server. Z toho vyplývá například problém se statickými třídami a jejich dědičností. Protože PHP 5.1 neumí tzv. late static binding[22], není možné volat statickou metodu rodičovské třídy, která pracuje se statickou proměnou z aktuální třídy. Statické proměnné jsou viditelné vždy v kontextu třídy, ve které je metoda implementována. Některé metody je nutné ve třídách reimplementovat, protože není možné použít jednoduchou dědičnost statických tříd.

Další nevýhodou PHP verze 5.1 je nemožnost používat vlastní definované namespace. Proto je třeba, aby se jednotlivé třídy jmenovaly vždy unikátně. Tudíž je v projektu použita struktura pojmenovávání podobná jako v Zend Frameworku 1.x. Tedy, že je každý nový namespace veden jako složka v souborovém systému a název třídy je hierarchicky vytvořen z názvů nadřazených složek a názvu souboru, oddělených podtržítkem.

Z důvodu podpory více jazyků webu FITkitu, je také v obou částech celého systému počítáno s možnými překlady. Všechny texty, které se vyskytují v obou aplikacích jsou volány přes funkce starající se jejich lokalizace. Samotné překladové řetězce jsou pak uloženy ve zvláštních souborech, aby je bylo kdykoliv možné snadno upravit. V obou případech se prozatím počítá pouze se angličtinou a češtinou, jako je tomu u současného webu FITkit.

Veškeré zdrojové kódy psané v jazyce PHP jsou rovněž ukládány v kódování CP1250, tedy standardním kódováním MS Windows. Je to především z důvodů, že server na kterém aplikace poběží, nabízí webové stránky právě v tomto kódování. Zdrojové soubory by mohly být v libovolném kódování, ale bylo by vždy nutné obsah překódovat do znakové sady předkládané uživateli. Stejným způsobem by bylo nutné převádět text zadaný uživatelem do formulářových prvků před uložením do databáze. Z tohoto důvodu bylo zvoleno kódování CP1250, aby se těmto zbytečným transformacím předešlo.

22 *PHP: Late Static Bindings - Manual* [online]. Poslední úpravy 18.5.2012 [cit. 19.5.2012]. Dostupný z WWW: <http://php.net/manual/en/language.oop5.late-static-bindings.php>.

8.1 Struktura webové aplikace

Implementovaná struktura souborů webové aplikace se snaží co nejvíce přiblížit současnému řešení webu projektu FITkit. I přesto, že se jedná o samostatnou aplikaci, která je nezávislá na zbytku systému, je možné ji nazvat modulem současného řešení. Struktura tedy vychází z navrženého modelu, kde je soubor *bugs_index.php*, který funguje jako tzv. kontrolér a řídí běh celého modulu. Jako první se v tomto souboru načte konfigurace celého modulu ze souboru *bugs_config.inc.php*, obsahující všechna nastavení potřebná pro běh aplikace. Následně se podle požadované akce načte obslužný skript z podsložky *inc*.

Speciální podsložkou složky *inc* je adresář *lang,* obsahující soubory s lokalizacemi všech textů použitých v aplikaci. Každý soubor s překlady má název tvaru *cs.php* a vypadá jako jednoduchá třída s jediným statickým atributem *words*, obsahujícím asociativní pole se všemi texty a jejich překlady. Součástí podsložky je navíc i pomocný skript *findTranslate.php* na hledání řetězců potřebných k překladu ve všech zdrojových kódech celé aplikace.

Všechny obslužné skripty se snaží být co nejjednodušší a nejpřehlednější. Jejich hlavní úlohou je zkontrolovat vstupní parametry, zareagovat na odeslané formuláře a zobrazovat HTML výstup nabízený uživateli. Aby byla co nejvíce udržena přehlednost, snaží se aplikace rozdělit co nejvíce funkčního kódu mezi objekty konkrétních tříd.

V konfiguračním skriptu je zadána i cesta pro vkládání všech potřebných tříd, která je uložena ve složce *library,* vedle skriptů z pomyslné prezentační vrstvy. Tato složka je, jak již bylo řečeno, hierarchicky uspořádána podle názvů jednotlivých tříd tak, aby byla každá třída fyzicky ve vlastním souboru. Zpřehlední se tím jak náležitost jednotlivých tříd do jednotlivých jmenných prostorů, tak i samotné hledání a vkládání správných tříd podle jejich názvů.

8.2 Práce s databází

Pro práci s připojením k databázi je v systému připravena třída *Database*, která vychází z návrhového vzoru singleton. I přes to, že je v PHP singleton pomalejší než čistě statická třída [13], je v tomto případě tento návrhový vzor vhodný. Zejména proto, že se vytvoří jediná instance třídy, která inicializuje připojení k databázi a neotvírá tak zbytečně připojení při každém dotazu na databázi. V rámci destruktoru této třídy se pak spojení vždy řádně ukončí.

Třída slouží především pro snazší práci s připojením k databázi a voláním dotazů nad ní, současně se stará o automatické ošetření řetězců ukládaných do databáze, aby předešla útokům typu SQL injection. Pokud by v budoucnu bylo třeba změnit databázový systém, není nutné přepisovat zdrojové kódy celého projektu, ale stačí upravit jen třídu pro práci s databází, která se snaží odstínit přístup k databázi od samotného volání ve zbytku aplikace.

Při jakékoliv chybě, ať už na straně databáze nebo při volání jednotlivých metod třídy, je vyvolána výjimka *Database_Exception* s informací o chybě v anglickém jazyce. Obecně jsou texty výjimek v programu psány pouze v angličtině. K výjimkám by mělo docházet jen sporadicky a je zbytečné kvůli tomu lokalizovat i všechna takováto chybová hlášení.

8.2.1 Manipulace s daty

Manipulaci s daty v databázi má na starosti rovněž třída *Database* obsahující několik jednoduchých metod, které řeší jak přistupovat k databázi. V první řadě je třeba se k databázi připojit, k čemuž slouží metoda *connect()*.

Pokud celý systém běží v jiném kódování než samotná databáze, je třeba nastavit správné připojení k databázi pomocí metody *setCharset()*. V případě spuštění systému na školním serveru, je třeba nastavit kódování připojení k databázi na již zmíněný formát CP1250.

Ošetření vstupů, aby nedocházelo k útokům typu SQL injection nebo cross-site scripting, má na starosti metoda *escape()*. Nejprve nahradí všechny nebezpečné znaky za odpovídající HTML entity a následně upraví řetězec pomocí funkce *mysql_real_escape_string()*. Díky tomu se předejde již zmiňovaným útokům. Pro volání konkrétních dotazů nad databází slouží prostá metoda *query()*, která obdobně jako funkce *mysql_query(),* volá dotaz nad aktuálně připojenou databází a vrací výsledek typu *resource*. Tato metoda je využívána i v ostatních metodách, které volají databázové dotazy. Pokud je zadaný dotaz typu INSERT a primární klíč položky je typu AUTO_INCREMENT, je možné využít metodu *lastId()*, která vrací identifikátor posledně vloženého záznamu.

8.2.2 Transakční zpracování

V některých situacích je třeba volat databázové dotazy v transakcích, aby se tak předešlo nekonzistentnímu stavu aplikace. Příkladem může být zápis informací do několika tabulek zároveň. Pokud by se data zapsala do první tabulky a u druhé by nastala chyba, je třeba vrátit záznamy v první tabulce do původního stavu. V opačném případě by byly záznamy v obou tabulkách nekonzistentní a mohlo by to znamenat neočekávané chování celé aplikace.

Pro práci s transakcemi obsahuje databázová třída základní metody na jejich řízení. Začátek transakce je vyvolán metodou *transactionStart()* a od této doby je transakce platná. Ukončení transakce se provádí voláním metody *transationCommit()*, která potvrdí celou transakci a uvede databázi do stavu po provedení všech požadovaných změn. Pokud během vykonávání příkazů dojde k chybě nebo nevhodnému stavu databáze, je možné ukončit celou transakci voláním metody *transactionRollback()*, která vrátí všechny provedené změny až do bodu, kde byla transakce spuštěna.

8.2.3 Získávání dat

Procházení výsledků dotazů nad databází je v PHP možné vždy jen po řádcích a není tak jednoduché získat všechny záznamy v jednom poli. Proto obsahuje třída *Database* několik metod na snazší práci s řádky načtenými z databáze.

Základem je metoda *fetchRow()*, která vrací pouze jeden řádek z databáze a je možné druhým nepovinným parametrem určit, zda se má vrácené pole indexovat asociativně nebo jen čísly, jako je tomu u standardních funkcí pro práci s MySQL databází. Ve výchozím stavu vrací metoda asociativní pole všech sloupečků pro výsledek předaného dotazu. Metoda *fetchColumn()* se chová obdobně, akorát nevrací celý řádek, nýbrž jen hodnotu jednoho daného sloupečku z něj. Číslo sloupce je možné předat jako jeden z parametrů. Pokud ale není žádné číslo zadáno, metoda vrací vždy hodnotu prvního sloupečku pro požadovaný dotaz.

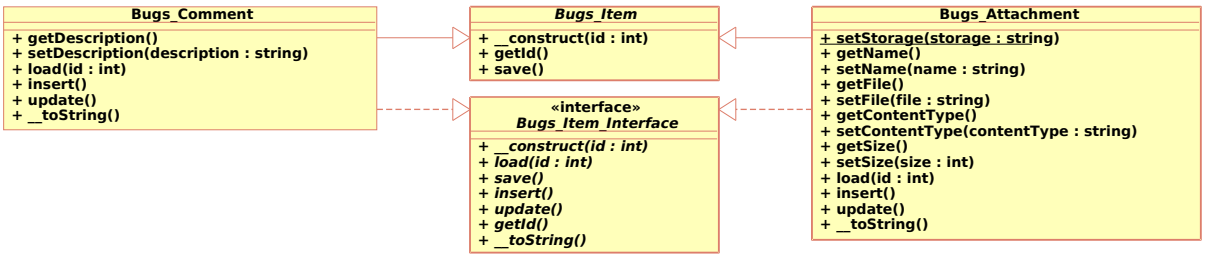
Získání všech záznamů v jednom PHP poli má na starosti několik metod, které jsou upraveny pro konkrétní potřeby. Nejjednodušší je metoda *fetchAll()* vracející dvourozměrné pole všech záznamů a všech jejich hodnot. Pomocí nepovinného parametru, je stejně jako u *fetchRow()* možné určit, zda se mají vrácené záznamy ukládat do indexovaného nebo asociativního pole. Výsledek tvoří pole záznamů, ve kterém jsou jednotlivé řádky řazeny tak, jak bylo specifikováno ve volaném dotazu. Oproti tomu metoda *fetchAllKey()* vrací asociativní pole, kde je index každého záznamu dán prvním sloupečkem volaného dotazu. Nejčastěji se jedná o primární klíč, takže je pak možné z tohoto pole jednoduše vybrat záznamy pro daný klíč. Na stejném principu funguje i poslední metoda pro načítání záznamů nazvaná *fetchAllPairs()*. Jejím výsledkem je pouze jednorozměrné pole a jeho vstupem je dotaz, který vrací minimálně dva sloupečky z databáze. První sloupeček se použije jako klíč ve výsledném poli a druhý jako jeho hodnota. Jedná se tak tedy o klasické dvourozměrné pole typu *key-value*. To je vhodné především u různých číselníků, kde si stačí pamatovat jen identifikátor a jeho textovou hodnotu.

8.2.4 Objektově relační mapování

U většiny dnešních frameworků je obvyklé, že využívají objektově relační mapování. Jedná se o technologii, kdy se záznamy z tabulek v databázi programově mapují na objekty tříd v PHP. Tento způsob je užitečný především svou abstrakcí v rámci tvořené aplikace, programátor tak pracuje s objekty dané třídy a nemusí přímo řešit jejich fyzické uložení do tabulek v databázi. Vhodné to je především pokud je u objektu potřeba vytvářet i vazby na jiné objekty a tyto pak při ukládání zohledňovat. Programátor je jen nastaví a už nemusí řešit, do jakých tabulek je třeba záznamy přidat.

Podobný zjednodušený systém, založený na návrhovém vzoru Active Record, popsaný v knize Patterns of Enterprise Application Architecture [14], je implementován i v tomto projektu. Jedná se o základní mapování databázových tabulek na jednoduché objekty obsahující atributy podle sloupečků v tabulkách. Základem je abstraktní třída *Bugs_Item*, která obsahuje základní metody společné pro všechny potomky této třídy. Jedná se především o konstruktor, který podle zadaného identifikátoru hledá odpovídající záznam v databázi a stará se o uložení instance databázového připojení přímo v konkrétním objektu. Stejně tak všechny tyto třídy implementují rozhraní *Bugs_Item_Interface()* sjednocující práci s těmito objekty v celé aplikaci. Podrobný diagram tříd, který ukazuje vztahy mezi všemi databázovými třídami je vidět v příloze F.

Každá třída musí rovněž implementovat tři základní metody pro manipulaci se záznamy v databázi. Metoda *save()* je implementovaná již v abstraktní třídě *Bugs_Item* a podle identifikátoru objektu se rozhodne, zda se do databáze objekt nově vloží nebo jen přepíše stávající záznam voláním dalších dvou metod *insert()* a *update()*. Tyto metody musí implementovat každá třída zvlášť, jak je vidět na následujícím obrázku 4. Jedná se o dvě konkrétní třídy pro manipulaci s přílohou a komentářem.



Obrázek 4: Závislost tříd Bugs_Item

Složitější objekty, obsahující záznamy z jiných tabulek, vždy obsahují i objekt pro tento záznam. Aby nedocházelo k nekonečnému zanoření těchto objektů, je většinou tato informace doplněna až na vyžádání. Jako je tomu třeba u historie konkrétního ticketu. Třída *Bugs_Ticket* obsahuje metodu *getHistory()*, která vrací pole všech položek historie. Protože není třeba znát celou historii pro každý nový ticket, není třeba ji generovat předem a zbytečně si ji pamatovat.

Některé třídy obsahují kromě metod na nastavení a získání jejich vlastností i některé specifické metody podle potřeby. Příkladem může být již zmíněná třída *Bugs_Ticket* nebo rozsáhlá třída *Bugs_Version*. Aby byla práce s objekty verzí co nejsnazší, obsahuje třída *Bugs_Version* několik specifických metod například pro změnu stavu verze. To především proto, že je nutné nejen změnit stav, ale rovnou na něj i systémově reagovat a odeslat například informační e-maily udržovatelům balíčků nebo správci.

Doposud byl pro každý záznam v databázi tvořen samostatný objekt, ale při práci se záznamy je třeba je i vyhledávat, případně filtrovat podle zadaných kritérií. Pro všechny tyto manipulace s větším počtem objektů stejného typu, je v systému navržena speciální abstraktní třída *Bugs_Table*, která obsahuje statické metody pro práci s více stejnými objekty nad databází. Ve vyšší verzi PHP by bylo možné navrhnout tyto statické metody tak, aby byly společné pro všechny potomky této třídy a chovaly se specificky podle konkrétního nastavení dané podtřídy. Bohužel verze PHP 5.1 nepodporuje tzv. late static binding a je tedy nutné toto chování simulovat jinak. Každá třída, která rozšiřuje třídu *Bugs_Table*, obsahuje název databázové tabulky a volá metody nadřazené třídy s tímto názvem. Pokud by metody a atributy tříd nebyly statické, stačilo by v hlavní třídě vytvořit pouze metodu, která by automaticky pracovala s atributem konkrétní třídy, jako je vidět v ukázce 1.

```
class Bugs_Table {
   protected $table = 'unknown';
   ...
   public function maxId() {
      $db = Database::getInstance();
      $sql = "SELECT max(id) FROM " . $this->table;
      return  $db->fetchColumn($sql);
   }
   ...
}

class Bugs_Versions extends Bugs_Table {
   protected $table = 'versions';
   ...
}
```

Ukázka 1: Nestatická třída Bugs_Table

Volání metody *maxId()* nad objektem třídy *Bugs_Versions* vrací maximální identifikátor z tabulky *versions*, nikoliv z tabulky *unknown*, což je požadované chování a většina vývojářů je na něj zvyklá.

U statického volání v PHP 5.1 je ovšem situace zcela odlišná. V případě statických metod a atributů, by po zavolání metody *maxId()* nad objektem třídy *Bugs_Versions* metoda vracela také maximální identifikátor, nikoliv však z tabulky *versions*, ale z tabulky definované v hlavní třídě, tedy *unknown*. Je to z důvodů, že PHP vyhodnocuje klíčové slovo *self* již při překladu. PHP od verze 5.3.0 zavádí speciální klíčové slovo *static*, které má těmto problémům předejít, protože vyhodnocuje toto volání až při samotném běhu.

Z těchto důvodů bylo nutné problém řešit jinak než pomocí klíčového slova *static*. Aby byl dodržen alespoň částečně navržený model statického volání, byl implementován kód zobrazený v ukázce 2. Tedy princip, kdy se v dané podtříd volá pouze metoda nadřazené třídy s parametrem, který určí konkrétní chování zvolené metody. Výsledkem je stejný princip jako u ukázky 1, ale volaný staticky.

```
class Bugs_Table {
   protected static $table = 'unknown';
   ...
   public static function maxId($table) {
      $db = Database::getInstance();
      $sql = "SELECT max(id) FROM " . $table;
      return $db->fetchColumn($sql);
   }
   ...
}

class Bugs_Versions extends Bugs_Table {
   protected static $table = 'versions';
   ...
   public static function maxId() {
      return parent::maxId(self::$table);
   }
   ...
}
```

Ukázka 2: Statická třída Bugs_Table

Statické volání je v tomto případě užitečnější, protože se jednotlivé metody chovají spíše jako pomocné funkce, než samostatné objekty. Není proto třeba při každém volání vytvářet novou instanci třídy, ale stačí zavolat její statickou metodu, vracející například pole požadovaných objektů.

Jako praktická ukázka může posloužit třída *Bugs_Tickets*, která obsahuje metodu *getAll()* vracející pole objektů typu *Bugs_Ticket* podle zadaných kriterií. Nebo zmiňovaná třída *Bugs_Versions* obsahující hned několik metod na vrácení seznamu verzí. Konkrétně se jedná o metody *getAll()* vracející všechny verze v systému, *getLast()* vracející poslední uzavřenou verzi nebo *getReleased()*, která vrací jen uzavřené verze.

8.3 Práce se Subversion

Podobně jako práce s databázovými tabulkami, je i procházení Subversion zabaleno do tříd. Všechny tyto třídy a jejich použití je podrobně popsáno v následujícím textu.

Základem je třída *Svn*, která se stará o získávání revizí a udržuje si vždy hodnotu poslední revize. V konfiguraci aplikace je třeba inicializovat tuto třídu jednou statickou metodou s předanou adresou repozitáře, se kterým se má pracovat. Systém si v databázi eviduje seznam všech revizí, včetně informací o autorovi, datu a připojené zprávě. Je to z důvodu, aby nebylo nutné neustále se dotazovat repozitáře na aktuální revizi. To ale znamená, pravidelně tuto tabulku aktualizovat. Nejlepším řešením by bylo aktualizovat hodnoty vždy při každém commitu nad daným repozitářem, čehož je možné docílit například napsáním jednoduchého post-commit hook scriptu. To je skript, který se vykoná vždy po provedení příkazu commit nad daným repozitářem. Bohužel tato možnost není na nasazovaném serveru dostupná a je třeba jí tedy řešit jinak.

Nabízí se aktualizovat hodnoty tabulky v pravidelných intervalech pomocí skriptu pro *cron*, ale to by znamenalo možné prodlevy mezi odesláním dat do repozitáře a další aktualizací. To není zcela žádoucí, především proto, že se dá předpokládat, že uživatel, který provedl commit, bude chtít s webovou aplikací okamžitě pracovat.

Hodnoty v databázi je proto nutné aktualizovat při každém načtení stránky. Znamená to provádět pokaždé minimálně jeden dotaz na samotný repozitář. Výhodou je, že se jedná jen o jednoduchý dotaz vracející pouze číslo poslední revize. Toto číslo se následně porovná s poslední hodnotou v databázi a pokud se liší, teprve tehdy je vyžádán seznam všech nových změn. O kontrolu poslední verze se stará metoda *checkHead()* zmíněné třídy *Svn*. Tato třída si následně udržuje statickou hodnotu čísla poslední revize, aby se na ni nebylo nutné za běhu skriptu neustále znovu dotazovat.

Kódování připojení k databázi je v rámci celé aplikace nastaveno na zmíněných CP1250, ale informace o jednotlivých revizích z repozitáře jsou v kódování UTF8. Proto je nutné před zápisem těchto informací do databáze, změnit aktuální kódování a po provedení změn jej opět vrátit zpět na původní hodnotu.

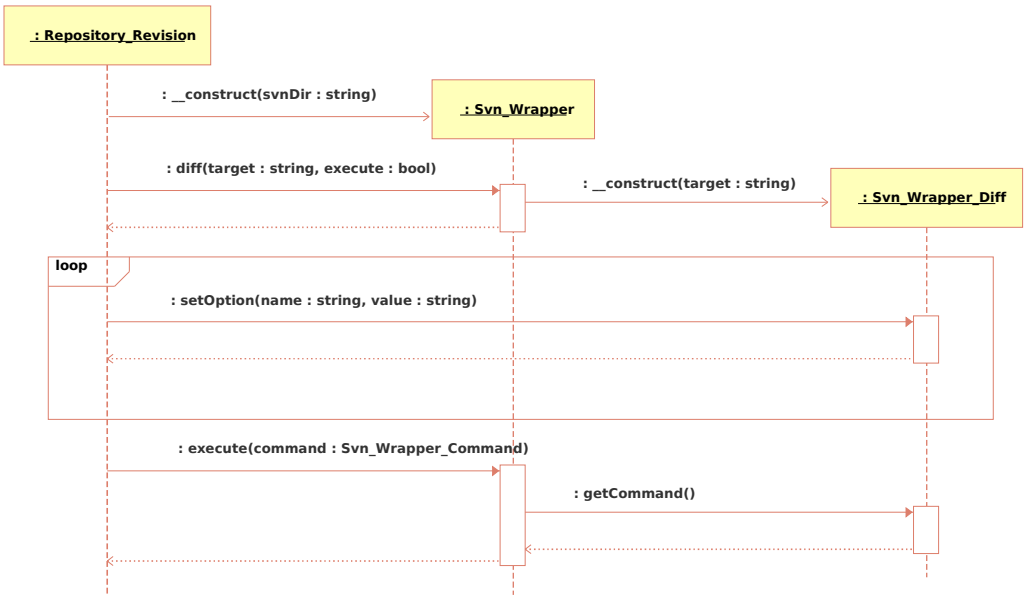
8.3.1 Volání příkazů svn

Příkazy *svn* je třeba volat pomocí PHP funkce *system()* a proto byl navržen princip, kdy se o samotné volání stará pouze jedna třída. Jedná se o třídu *Svn_Wrapper*, která obsahuje metody pro všechny implementované příkazy *svn*. Pro každý příkaz existuje samostatná třída zděděná od třídy *Svn_Wrapper_Command*. Součástí všech těchto tříd jsou tři povinné metody částečně implementované už v rodičovské třídě *Svn_Wrapper_Command*. První je metoda *setOption()* sloužící k nastavení voleb daného příkazu. Povolené volby jsou uloženy v atributu *allowed_options* jednotlivých tříd. Druhá metoda *getCommand()* má za úkol vrátit řetězec příkazu poskládaný ze všech předem zadaných voleb. Poslední metoda *getName()* pouze vrací název dané třídy.

Princip volání příkazů přes obalující třídu *Svn_Wrapper* je dobře vidět na obrázku 5. Nejprve se vytvoří instance třídy *Svn_Wrapper*, která v sobě vytvoří instanci třídy požadovaného příkazu. Tomu se následně nastaví potřebné volby metodou *setOption()* a požádá se o vrácení řetězce pro volání *svn* příkazu. Objekt třídy *Svn_Wrapper* buď následně provede příkaz sám, pokud je předán parametr *execute* při volání metody nebo čeká na zavolání metody *execute()* nadřazeným objektem, tak jak je tomu v zmiňovaném příkladu.

Metody třídy *Svn_Wrapper* i všech tříd jednotlivých příkazů, které nevrací konkrétní hodnoty, se snaží splňovat tzv. fluent interface. Tedy, že vrací samy sebe, aby bylo možné volat jednotlivé metody jednoduše za sebou bez nutností neustále opakovat název objektu. Tento způsob je v případě nastavování různých parametrů ve zdrojovém kódu aplikace daleko přehlednější. Příklad takového volání je názorně vidět na ukázce 3, kde se nejprve vytvoří objekt třídy *Svn_Wrapper*, který při volání metody *diff()* vrací objekt třídy *Svn_Wrapper_Diff*. Tomuto objektu se následně nastaví potřebné volby.

Pro volání všech implementovaných příkazů jsou vytvořeny automatizované testy ve formátu PHPUnit. Díky nim je možné snadno kontrolovat, zda se všechny příkazy volají a provádějí tak, jak je očekáváno a že žádný z nich nezpůsobuje při svém volání chybu.



Obrázek 5: Volání metod objektu Svn_Wrapper_Diff

```
$svn = new Svn_Wrapper();
$svn->diff(Svn::$target)
    ->setOption('xml')
    ->setOption('summarize')
    ->setOption('change', $this->revision);
$svn->execute();
```

Ukázka 3: Volání metod objektu Svn_Wrapper_Diff

8.3.2 Vyrovnávací paměť

Volání příkazu *svn* není příliš rychlé a je tedy vhodné jej neprovádět při každém dotazu na repozitář. Z tohoto důvodu byl implementován přístup ukládání výsledků volaných příkazů do souborů v souborovém systému. U systému s velkou návštěvností by bylo možné tyto data ukládat například i do paměti RAM daného serveru, aby práce s daty byla co nejrychlejší, ale to není v této situaci nutné.

Před každým voláním příkazu, se tento předá metodě *load()* třídy *Svn_Wrapper_Cache*, která se pokusí najít ve vyrovnávací paměti dříve uložený výsledek. Z každého předaného příkazu se pomocí PHP funkce *md5()* vytvoří unikátní hash určující soubor ve vyrovnávací paměti. Pokud je soubor s vytvořeným hashem již v paměti, změní se čas modifikace souboru a vrátí jeho obsah. Změna času modifikace se děje z důvodu, aby bylo, například při údržbě, jednoduše možné určit, se kterými soubory se dlouho nepracovalo a je možné je z vyrovnávací paměti na disku odstranit. Pokud soubor na disku zatím není, zavolá se příkaz běžným způsobem a výsledek se zapíše do cache voláním statické metody *save()* třídy *Svn_Wrapper_Cache*. Soubory se ve vyrovnávací paměti ukládají do podsložek podle typu příkazu. Souborová struktura je tak přehlednější a je možné smazat soubory například pouze pro příkaz *Svn_Wrapper_Diff*.

Metodou *setCompress()* se povoluje komprese souborů ukládaných do vyrovnávací paměti, čímž se šetří místo na cílovém disku. Většina souborů je totiž pouze textová a proto je vhodné je před uložením komprimovat. Při čtení se opět zpětně provádí dekomprimace, takže jsou data uživateli servírována tak, jak by je vrátilo samotné volání příkazu.

Před požitím vyrovnávací paměti, je nutné provést prvotní nastavení v souboru *bugs_config.inc.php*. Jedná se především o nastavení cesty k adresáři s vyrovnávací pamětí, pomocí metody *setDir()*, aby systém věděl, kam soubory ukládat. Dále je možné metodou *setCompress()* určit, zda se mají data před uložením na disk komprimovat. A v neposlední řadě je třeba celou vyrovnávací paměť povolit metodou *enable()*.

8.4 Procházení revizí a souborů

Část starající se o procházení souborů uložených v repozitáři je rozdělena na do tří akcí a jim odpovídajících tříd. Každá třída řeší trochu jinou část práce s repozitářem.

Procházení složek v repozitáři má na starosti třída *Repository_Browser*. Její účel je zřejmý, vypisuje soubory a podsložky zvolené složky v předané revizi. Seznam vypisovaných souborů je ve formě PHP pole obsahujícího objekty třídy *Repository_File* pro každou položku. Navíc se třída stará o výpis drobečkové navigace umožňující návrat v adresářové struktuře repozitáře. Pro každou vybranou složku zároveň vypisuje všechny její nastavené vlastnosti. Například vlastnost *svn:ignore*, určující jaké soubory ve složce jsou ignorovány při nahrávání na server.

Výpis obsahu souborů je řešen třídou *Repository_File*. Obsah souboru je většinou textový a proto je možné ho přímo zobrazit uživateli. Byl by ale lehce nepřehledný a je vhodné ho před zobrazením ještě naformátovat. K tomu je v projektu použita externí třída *Geshi*[23], pro podbarvování

23 *GeSHi - Generic Syntax Highlighter :: Home* [online]. Poslední úpravy 11.2.2011 [cit. 1.5.2012]. Dostupný z WWW: <http://qbnz.com/highlighter>.

syntaxe různých programovacích jazyků. Text souborů je tímto způsobem přehledně zvýrazněn. Třída *Geshi* umí zpracovat velké množství formátů a je tak pro tento projekt více než vhodná. Binární soubory není tímto způsobem možné zobrazovat a tak nejsou vypisovány vůbec. Jedinou výjimkou jsou obrázky typu jpg, gif a png. Ty jsou místo textového zobrazení rovnou vykresleny v prohlížeči. Každá instance třídy *Repository_File* navíc obsahuje podrobné informace o typu souboru, velikosti, autorovi, nebo například ikoně, která se má zobrazovat ve výpisu souborů. Stejně jako u výpisu souborů a složek i v tomto případě obsahuje třída položku s vlastnostmi vztahujícími se k danému souboru.

Poslední třída se stará o vrácení všech souborů změněných v určité revizi. Zároveň vrací i informace o dané revizi jako je autor revize, zpráva při commitu, datum nebo například stáří revize, pro zobrazení ve výpisu souborů a složek. Ke každému soubor navíc připojuje informaci o změnách jednotlivých řádků souboru, které navíc zvýrazní. Rozlišuje tím smazané řádky od těch nově přidaných.

8.5 Autorizace uživatelů

Autentizaci uživatelů řeší současný webový systém FITkit automaticky, takže není nutné řešit autentizaci v implementované aplikaci explicitně, je ale nutné přihlášené i nepřihlášené uživatele autorizovat k provádění požadovaných operací.

Systém pracuje prakticky se čtyřmi typy rolí i přesto, že v databázi se udržují pouze dvě: administrátor a udržovatelé balíčků. Zbylé dvě role slouží k odlišení všech přihlášených uživatelů od nepřihlášených návštěvníků webu. Rozlišení první role od zbylých tří je snadné, stačí pouze kontrolovat systémovou proměnou informující o tom, zda je uživatel přihlášen či nikoliv. Problém nastává až u odlišení tří nejvyšších oprávnění. Je nutné je systémově kontrolovat podle jednotlivých přihlášených uživatelů.

Autorizace jednotlivých uživatelů je založena na jednoduchém principu, kdy se při každém přístupu k systému vytvoří nový objekt třídy *Bugs_User*. Tomu se v konstruktoru předá login přihlášeného uživatele a objekt se tak automaticky inicializuje daty z databáze. Pokud uživatel s předaným loginem v databázi není, nebo není přihlášen, vytvoří se nový prázdný objekt této třídy. Objekt třídy *Bugs_User* vždy obsahuje speciální metodu *checkRole()*, která slouží k ověřování oprávnění vůči přihlášenému uživateli. Její první parametr je textová reprezentace role v systému, shodná s definicí rolí v databázové tabulce *users*. Vůči této roli se kontroluje role uživatele daného objektu. Metoda vrací *True*, pokud je role uživatele minimálně tak velká jako požadovaná role, v opačném případě vrací *False*, což znamená, že daný uživatel nemá dostatečná oprávnění.

8.6 Správa ticketů a verzí

Přidávání a procházení ticketů a verzí je spjato s konkrétními databázovými třídami. Ty jsou rozšířeny o specifické metody pro manipulací jednotlivých objektů a změny jejich vlastností. Aplikace se chová pouze tak, že například při změně upraví odpovídající údaje ve vytvořeném databázovém objektu a tento objekt uloží. Tím se data automaticky uloží i do správných databázových tabulek.

Změna stavu ticketu tak, jak bylo popsáno v návrhu, je částečně automatická a částečně řešena doplněním vhodných tlačítek a na ně navázaných akcí. Při změně vlastníka ticketu se mění stav ticketu na hodnotu *„přiřazeno“*. V tento moment se administrátorovi a vlastníkovi ticketu nabízí možnost uzavřít ticket. Poté je administrátorovi umožněno ticket znovu otevřít a následně opět uzavřít stejným postupem. Všechny změny stavu ticketu se dějí přes volání metody *setStatus()* objektu třídy *Bugs_History*. Podmínky, kdy se má měnit stav jsou definovaný přímo v úvodu souboru *ticket.inc.php*, který se stará o zobrazení i ukládání nastavených vlastností ticketu.

Životní cyklus verzí, jak byl popsán v návrhu, je v systému implementován podobně jako u ticketů. Administrátor má možnost v detailu verze měnit její stav. Nikoliv však ručně, ale pomocí postupného potvrzování jednotlivých stavů. Protože je při změnách stavů verze třeba dělat i další činnosti, je v rámci třídy *Bugs_Version* implementováno i několik specifických metod. Jedná se především o metodu *setStatus()*, která v sobě automaticky kontroluje, zda jsou splněny všechny podmínky pro změnu stavu. Například při požadavku na připravení verze k uzavření, je třeba, aby byly uzavřeny všechny tickety v ní. Při uzavírání musí být verze pro změnu potvrzená od všech účastníků. Jedinou automatizací při změně stavů je situace, kdy všichni udržovatelé potvrdí, že je verze připravena. V tento moment se automaticky změní stav na hodnotu *„připraveno k uzavření“*.

Tato třída zároveň obsahuje několik konkrétních funkcí na změny stavu, které nejen že mění tento stav, ale navíc vykonávají i jiné, s tím spojené úkoly. Především zasílají e-maily všem, do procesu zapojeným uživatelům systému. K odesílání zpráv slouží jednoduchá třída *Bugs_Mail*, která má kompletně na starosti zasílání e-mailů zadaným adresátům. Pro každý typ zprávy obsahuje speciální metodu starající se o formát a obsah sdělení. Samotné odesílání je řešeno pomocí privátní metody *send()* volané z těchto jednotlivých metod. Odesílané e-maily není třeba přehnaně formátovat a tak stačí zasílat pouze textové zprávy, bez nutnosti složitě skládat tělo e-mailu z několika částí různých typů.

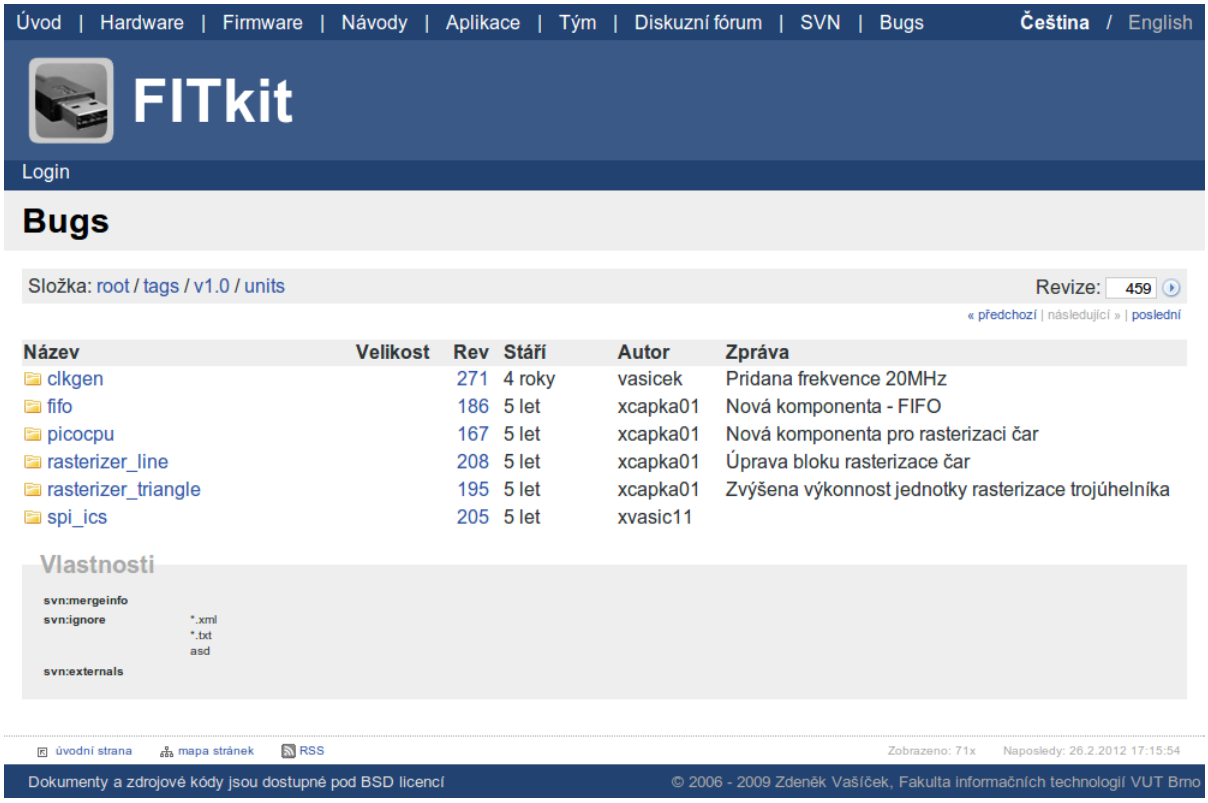
Třída *Bugs_Version* obsahuje i několik metod pro práci s udržovateli balíčků, kteří se na uzavření podílí. Jedná se o metody *addParticipant()*, *delParticipant()* pro ruční přidávání a odebírání udržovatelů a *addParticipants()* pro hromadné přidání. Poslední metoda je využita převážně při vytváření nové verze. K takto vytvořené verzi jsou automaticky doplnění všichni aktivní

udržovatelé balíčků, aby je administrátor nemusel pokaždé přidávat ručně. Všechny tyto změny se dějí v rámci skriptu *version.inc.php*.

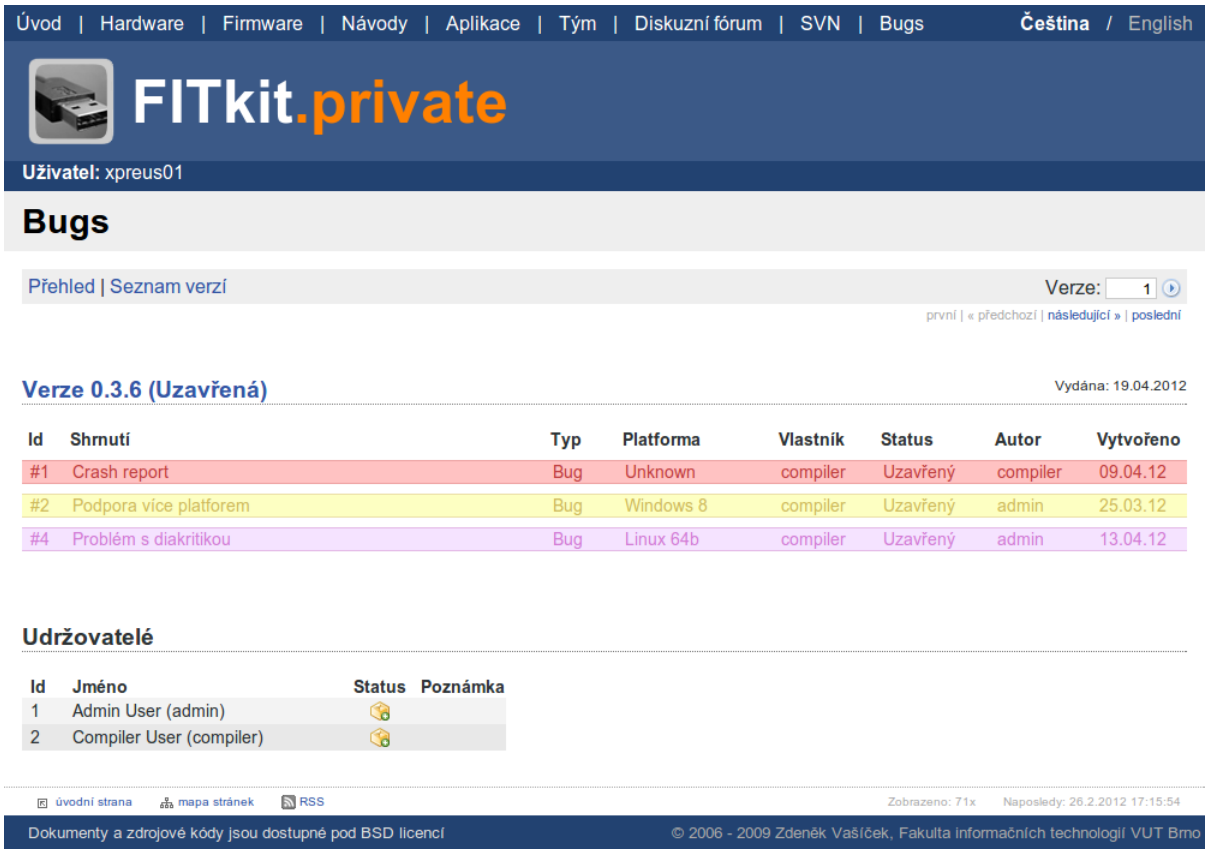
Pro snazší práci s HTML prvky typu SELECT, obsahuje aplikace speciální třídu *Bugs_Html*, která implementuje metodu *options(),* starající se o vykreslení jednotlivých položek prvku SELECT. Její výhodou je, že jí stačí předat pole položek s klíčí pro vykreslení a v případě potřeby i předvybranou hodnotu. Díky tomu, že všechny třídy odvozené od třídy *Bugs_Table* obsahují metodu *getArray()* na vrácení požadovaného pole, je možné snadno vypsat prvek SELECT s hodnotami různých číselníků z databáze.

8.7 Klientská část

Webová aplikace je rozdělena na dvě základní části, jedna pracuje s repozitářem a zobrazuje jeho obsah uživateli, jak je vidět na obrázku 6, druhá se stará o správu chyb a požadavků, stejně jako o jejich přiřazení k jednotlivým verzím aplikace a je zobrazena na obrázku 7. Návaznost jednotlivých obrazovek obou částí je znázorněna na diagramech v příloze G a H.



Obrázek 6: Obsah repozitáře



Obrázek 7: Detail verze

8.7.1 Manipulace s DOM strukturou

Současný web projektu FITkit obsahuje pouze jednoduché JavaScriptové funkce pro konkrétní účely, ovšem neobsahuje žádnou z hotových knihoven pro usnadnění práce, jako je například jQuery. Tyto knihovny obecně zahrnují mnoho funkcí, které nebyly pro tento projekt potřebné a tak by bylo zbytečné je využívat. Proto bylo v klientské části třeba napsat i několik jednoduchých JavaScriptových funkcí na práci s DOM strukturou stránky. Ale vždy jen to takové míry, která je pro projekt nezbytná.

Základem je objekt *dom* obsahující několik metod na manipulaci se strukturou stránky, implementované v čistém JavaScriptu, aby fungovaly na všech současných prohlížečích. Jedná se především o metody na vyhledávání elementů v DOM struktuře. Metoda *findByTag()* vrací první nalezený element, jež je zadaného typu. Výhodou všech implementovaných metod je, že lze druhým parametrem omezit prostor určený pro hledání. Díky tomu je prohledávání rychlejší a především se znalostí struktury stránky, může být využito k dohledání potřebných i opakujících se prvků.

Obdobně se chová i metoda *findByClass()*, která hledá první element zadané CSS třídy. Metoda *findAllByClass()* oproti předchozím nevrací pouze jeden výsledek, ale všechny nalezené prvky v předaném kontextu.

Dalším skupinou metod objektu *dom* jsou metody na práci s textem v celé struktuře. Metoda *getText()* vyhledá a vrátí prostý text předaného elementu a všech jejich potomků. Chová se tedy tak, že rekurzivně odstraňuje všechny HTML značky ze zadaného kontextu a výsledek vrací jako textový řetězec. Naopak metoda *replaceText()* prohledá všechny potomky zadaného prvku a nahradí hledanou podmínku, reprezentovanou regulárním výrazem, textem předaným v posledním parametru. Toho je využíváno především k zaktivnění odkazů provazující jednotlivé revize a tickety na celém webu.

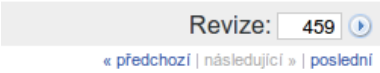
Při vkládání a volání JavaScriptových funkcí je vždy problém, kdy funkci deklarovat a kdy ji volat. Funkce pracující s DOM strukturou je ideální volat až když je celá struktura načtena, ale přehlednější je deklarovat je hned v úvodu nebo v externím souboru. Knihovna jQuery přichází v tomto případě se zajímavým řešením, jehož hlavní myšlenka byla použita i v tomto projektu. Součástí objektu *dom* je i speciální metoda *onReady()* čekající až bude načtena celá struktura stránky. Volání jednotlivých funkcí je vloženo v kontextu anonymní funkce, předaná metodě *onReady()* jako parametr. Metoda *onReady()* na pozadí čeká na načtení celé stránky a následně zavolá předanou anonymní funkci a provede celý její obsah.

8.7.2 Procházení repozitáře

Úvodní stránka první části zobrazuje výpis souborů kořenové složky v poslední revizi. Kliknutím na zobrazenou složku je možné procházet celým repozitářem, jak je tomu u běžných souborových manažerů. Zanoření ve složkách přehledně zobrazuje drobečková navigace v záhlaví stránky, jak je vidět na obrázku 8. Pomocí ní se lze kdykoliv vrátit do nadřazených složek.

Obrázek 8: Drobečková navigace

Podobně lze kdykoliv změnit číslo revize, ke které se soubory a složky vypisují. K tou slouží navigace v pravém rohu záhlaví stránky, zobrazené na obrázku 9. Je možné ručně zadat číslo požadované revize, nebo se postupně přepínat mezi sousedními revizemi.



Obrázek 9: Číslo revize

Ve výpisu souboru se dá přepínat mezi formátovaným textem pomocí knihovny *Geshi* a nebo prostým textem, který se může hodit například při kopírování do stránky. Přepínání je řešeno zcela pomocí navrženého objektu *dom* v JavaScriptu. Stránka je ze serveru posílána kompletně naformátována a po jejím celkovém načtení, se pomocí metody *getText()* odstraní všechno formátování. Server by mohl klientovi zasílat obě verze najednou, ale to by u velkých souborů znamenalo zbytečně příliš mnoho dat, která se dají jednoduše získat až na straně klienta. K přepínání slouží nabídka nad horní hranou výpisu obsahu souboru. Rozdíl formátovaného a prostého textu znázorňuje obrázek 10.

Formátovaný

```
#include <fitkitlib.h>
#include "audio_codec.h"

/*********************************
 * function sends codec configurati
 *********************************
void codec_SPI_write(unsigned int ad
{
  char dataH, dataL;
  dataH = (addr << 1) | ((data >> 8)
  dataL = data & 0xFF;

  /*
  term_send_str("Sending to Codec: "
```

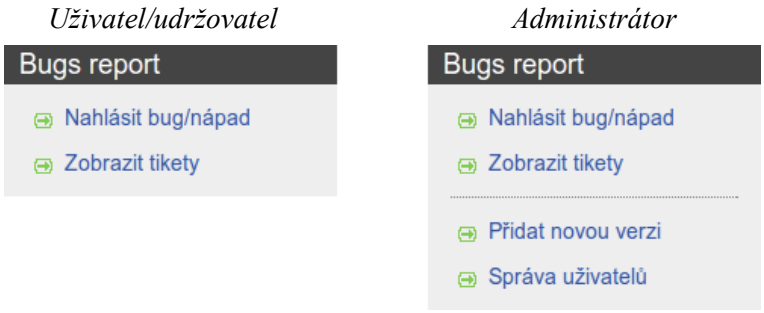
Neformátovaný

```
#include <fitkitlib.h>
#include "audio_codec.h"

/*********************************
 * function sends codec configurati
 *********************************
void codec_SPI_write(unsigned int ad
{
  char dataH, dataL;
  dataH = (addr << 1) | ((data >> 8)
  dataL = data & 0xFF;

  /*
  term_send_str("Sending to Codec: "
```

Obrázek 10: Rozdíly ve formátování

8.7.3 Správa chyb a požadavků

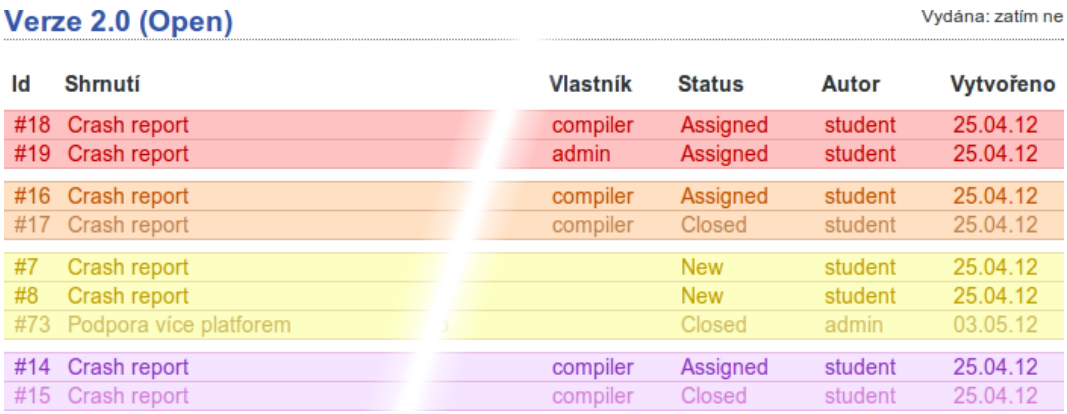
Stránka s přehledem části na správu chyb a požadavků se liší podle role uživatele pracujícího se sytémem. Rozdíl je pouze v kontextovém menu zobrazující akce, které může uživatel provádět. Nepřihlášený uživatel nevidí žádné menu a nemůže tak dělat nic víc, než procházet seznam verzí. Přihlášeným uživatelům se zobrazí menu naznačené na obrázku 11, podle role, kterou v rámci celé aplikace mají.



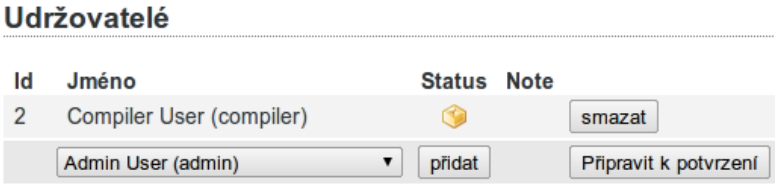
Obrázek 11: Základní menu

Výpis verzí zobrazuje jednotlivé verze řazené chronologicky podle data vydání. Aby nedocházelo ke zbytečnému načítání všech verzí, je nastaveno stránkování výpisu po šesti položkách. K přepínání mezi jednotlivými stránkami slouží navigace podobná té z přepínaní revizí při procházení repozitáře. Je opět v pravém horním rohu a lze jí ručně zadat požadovanou stránku nebo postupně mezi stránkami přepínat. U každé verze se vypisuje seznam změn, provedených v této verzi. Tento sezname neobsahuje všechny tickety přiřazené k verzi, ale pouze ty, které mají nastaven speciální text, tzv. release info. Bylo by nepřehledné vypisovat všechny tickety a proto se vypisují pouze ty, co jsou zajímavé.

Detailní zobrazení verze obsahuje seznam všech ticketů k ní přiřazených. V pravém horním rohu je opět navigace pro přepínaní mezi verzemi, jako je tomu u stránkování verzí. V levé části je navigace směřující uživatele zpět na celý přehled nebo do seznamu verzí. Výpis ticketů je v tomto případě řešen tabulkou, kde jsou jednotlivé řádky podbarveny podle závažnosti a stavu ve kterém se momentálně nachází. Uzavřené tickety jsou vždy méně výrazné, aby na první pohled vynikly tickety vyžadující další zásah správce nebo udržovatelů balíčků. Zkrácený výpis ticketů je vidět na obrázku 12. Uživatel s rolí alespoň udržovatele balíčků, navíc pod seznamem ticketů vidí i výpis všech udržovatelů podílejících se na verzi. Administrátor má navíc možnost jednotlivé udržovatele přidávat a odebírat podle situace. Současně v tomto místě udržovatelé verze potvrzují a administrátor je uzavírá pomocí nabízených tlačítek. Ta se zobrazují vždy jen v případě, že se verze nachází ve správném stavu a uživatel má právo verzi potvrzovat nebo uzavírat. Zobrazení udržovatelů z pohledu administrátora znázorňuje obrázek 13.



Obrázek 12: Zobrazení ticketů



Obrázek 13: Seznam udržovatelů

Náhled ticketu nabízí detailní informace o zvoleném ticketu, ale i jeho kompletní historii za celou dobu jeho životního cyklu, jak je vidět na obrázku 14. Nepřihlášený uživatel vidí pouze tyto informace. Přihlášenému uživateli se navíc zobrazí i možnost ticket měnit. Obyčejný uživatel může ticket pouze okomentovat, případně k němu přiložit libovolný soubor jako přílohu. Udržovatelé a administrátor mají navíc možnost měnit závažnost, vlastníka nebo verzi, do které ticket spadá. Rovněž mohou ticket nejen uložit, ale i uzavřít nabízeným tlačítkem. Uzavřený ticket mohou opět znovu otevřít podobným tlačítkem.

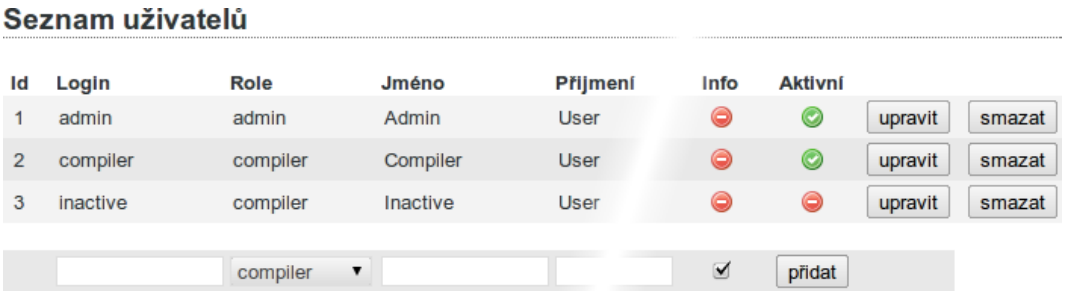
Formulář pro přidání nového ticketu je velmi obdobný formuláři pro editaci ticketu zmiňovaném v předchozím odstavci, jen s tím rozdílem, že neobsahuje tolik voleb. Je totiž univerzální pro všechny přihlášené uživatele. Předpokladem je, že bude sloužit především běžným uživatelům a není tak třeba jej zbytečně upravovat pro ostatní vyšší role. Položka s volbou platformy je automaticky předvybrána podle detekovaného operačního systému. O to se stará jednoduchá JavaScriptová funkce hledající potřebné informace v hodnotě *user-agent* uživatelova prohlížeče.

Obrázek 14: Zobrazení historie

Po uložení ticketu je uživatel automaticky přesměrován na detailní výpis. Tam mohou uživatelé s vyšší rolí zvolit dodatečné možnosti a ticket znovu uložit. Pro přesměrování je použit opět JavaScript, i když to není zrovna nejvhodnější způsob. Bohužel vychází ze současného navržení struktury aplikace, kdy není možné přesměrovat provádění skriptu pomocí hlavičky. Důvodem je, že se kód jednotlivých akcí vkládá do již vygenerovaného obsahu a v ten moment už není možné v PHP modifikovat hlavičku stránky nutnou pro přesměrování.

Systém umožňuje zobrazit tickety všech verzí na jednom místě. Je to především proto, aby bylo možné vidět i tickety nepřiřazené žádné verzi. Zároveň se tak snadno zobrazují všechny tickety ze všech verzí, přiřazené danému uživateli, což je vhodné pro všechny udržovatele balíčků i samotného administrátora. Těmto dvěma rolím se navíc při příchodu na stránku automaticky vyberou jen ty tickety, které ještě nejsou uzavřené a jsou přiřazeny právě jim. Tickety je možné dále filtrovat podle všech důležitých vlastností a všechna tato omezení výpisu libovolně kombinovat.

Administrátor má, jak bylo vidět na obrázku 10, dvě další možnosti jak s aplikací pracovat. Muže jednoduše přidávat nové verze systému a především spravovat uživatele, kteří mají vyšší práva. Editace i přidávání probíhají v rámci stejné stránky, aby bylo vše co nejjednodušší. Kliknutím na tlačítko upravit je možné změnit hodnoty libovolného uživatel a následně je uložit nebo zrušit. Informační ikonky znázorněné na obrázku 15, navíc umožňují rychlou změnu stavu, aby nebylo třeba přepínat na editaci položky.



Obrázek 15: Seznam uživatelů

8.8 QDevKit-bugs

Součástí sytému je možnost reportování chybových hlášení a pádů aplikace QDevKit. K odchytávání a následnému reportování těchto hlášení se používá knihovna Breakpad. To ovšem znamená upravit stávající aplikaci QDevKit tak, aby tuto knihovnu využívala a přitom byla i nadále jednoduše přeložitelná na požadovaných platformách.

Breakpad je navržen tak, aby byl pro použití co nejjednodušší. Vzhledem k jednotnému API stačí do programu vložit hlavičkový soubor a do hlavního těla připsat krátký handler starající se o zachytávání pádů a různých chybových stavů. Tento handler při chybě volá uživatelem definovanou funkci a zároveň vytvoří dump soubor s informacemi o pádu. Volaná funkce by měla dělat co nejméně úkonů, protože se tou dobou může program nacházet v nekonzistentním stavu. Proto v tomto případě pouze provede volání externí aplikace QDevKit-bugs se dvěma parametry, cestou ke složce, kde je uložen dump soubor a jeho názvem. Volání funkce i její obsah je nutné uzpůsobit konkrétním platformám, jak je vidět v ukázce 4.

Z důvodu zavedení závislosti na knihovně Breakpad bylo nutné upravit skripty k překladu aplikace QDevKit. Před kompilací aplikace je nutné mít v systému již připravený Breakpad. Automatický překlad pomocí nástroje *cmake* byl rozšířen o tuto novou závislost a bez ní aplikaci QDevKit nedovolí přeložit. Nástroj *cmake* hledá Breakpad v běžných systémových cestách, takže se může stát, že jej nenajde, pokud ho uživatel nainstalovat do neobvyklého umístění. V tom případě je před samotným překladem potřeba manuálně nastavit speciální proměnou prostředí *GOOGLE_BREAKPAD_DIR*.

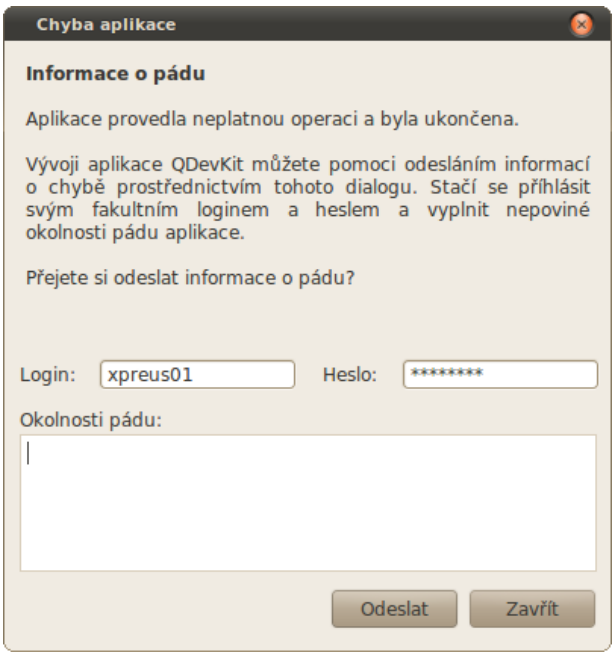
```
char * bugsPath;

#ifdef __WIN32
static bool DumpCallback(const wchar_t* dump_path, const wchar_t* minidump_id,
                         void* context, EXCEPTION_POINTERS* exinfo,
                         MDRawAssertionInfo* assertion, bool succeeded)
{
   _wexecl((wchar_t*) bugsPath, (wchar_t*) bugsPath, dump_path, minidump_id, NULL);
   return succeeded;
}
#else
static bool dumpCallback(const char* dump_path, const char* minidump_id, void* context,
                         bool succeeded)
{
   execl(bugsPath, bugsPath, dump_path, minidump_id, (char *) 0);
   return succeeded;
}
#endif
```

Ukázka 4: Funkce na zachycení chybových stavů

Aplikace QDevKit-bugs je napsaná v jazyce C++ s využitím knihoven Qt a přebírá z příkazové řádky dva vstupní parametry udávající, kde se nachází dump soubor. Pokud je cesta k souboru neplatná nebo je jeho velikost nulová, aplikace se hned ukončí, protože není třeba takovýto soubor zpracovávat a odesílat na server.

Z pohledu uživatele je aplikace velmi jednoduchá, jedná se pouze o jeden dialog zobrazený na obrázku 16. Pokud uživatel nechce pomoci vývoji aplikace QDevKit, může dialog zavřít a tím i smazat dump soubor z dočasného umístění. V opačném případě musí vyplnit školní login a heslo pro ověření vůči fakultnímu serveru.

Při kliknutí na tlačítko odeslat se na server posílají jak vyplněné informace, tak především dump soubor, vygenerovaný při pádu aplikace QDevKit. Aby nebylo nutné řešit na straně serveru speciální zachycení zasílaných informací, ale stačilo využít stávajícího principu, všechna data z dialogu se odesílají na server pomocí běžného HTTP POST požadavku. Princip je stejný, jako by byl požadavek odeslán přes webový formulář.



Obrázek 16: Chybový dialog

Školní server na kterém bude nasazena webová část aplikace, je přístupný prostřednictvím zabezpečeného protokolu HTTPS, což znamená, že aplikace musí zpracovat zasílané certifikáty. Kontrolu certifikátů má na starosti framework Qt, ale ten vyhodnotí školní certifikát jako nedůvěryhodný. Je proto nutné zachytit případné upozornění na podezřelý certifikát, aby nedošlo k přerušení spojení se serverem. Správné by bylo zobrazit upozornění uživateli, ale to by působilo rušivě a uživatel by ve většině případů jen stěží rozeznal problém s certifikátem školního serveru a nebezpečným certifikátem podstrčeným třetí stranou. Z tohoto důvodu je práce s certifikáty uživateli úplně skryta a aplikace ji řeší sama.

Aplikace taktéž neřeší problémy na straně serveru nebo například chybějící hodnotu při odesílání dat na server. Ve většině případů by na ni uživatel stejně nedokázal reagovat a je zbytečné ho s tím po pádu aplikace zatěžovat. Po odpovědi od serveru aplikace uživateli poděkuje a sama se ukončí.

V aplikaci je nutné řešit HTTP autentizaci oproti školnímu serveru, jako je tomu u přístupu k webovým stránkám pomocí prohlížeče. Autentizace je řešena pomocí třídy *QNetworkAccessManager* z frameworku Qt, které stačí předat jméno a heslo a ona se postará o výměnu informací se serverem. Pokud jsou zadané údaje neplatné, je na to uživatel upozorněn a požádán o jejich ověření.

Větší problém nastává u odesílání dat pomocí POST požadavku. Qt nenabízí kombinaci autentizace a posílání souborů a bylo tak třeba jednu z věcí implementovat ručně. Před odesláním dat na server se proto nejprve vytvoří řetězec obsahující tzv. multipart požadavek, se všemi potřebnými proměnnými a dump souborem tak, jak je popsán na stránkách W3C [15]. Takto sestavený dotaz se odešle na server pomocí metody *post()*. Požadavek je následně na straně serveru zachycen a PHP jej může zpracovat a ihned na něj reagovat.

Všechny texty aplikace jsou kompletně lokalizované do českého a anglického jazyka a stejně jako v aplikaci QDevKit, se přepínají podle aktuálního nastavení operačního systému uživatele. Primárně jsou všechny texty v angličtině, ale pokud uživatel pracuje v systému s českou lokalizací, aplikace se zobrazí rovnou v češtině, aniž by musel cokoliv nastavovat.

Aplikace QDevKit-bugs je multiplatformní a je tedy možné ji spustit na různých operačních systémech.

9 Závěr

V diplomové práci byly podrobně rozebrány současné možnosti nástrojů pro evidenci chyb a požadavků a bylo naznačeno jejich využití u projektu FITKit. Konfrontací existujících řešení s požadavky na navrhovaný systém bylo ukázáno, proč je nutné navrhnout a implementovat celý systém od základů.

Na základě analýzy existujících nástrojů na správu chyb a požadavků byl postupně navržen systém vhodný pro projekt FITkit uzpůsobený tak, aby jej bylo možné nasadit na školním serveru a aby zapadal do stávajících stránek projektu FITkit. Takové rozhodnutí s sebou nese určité kompromisy, ale i jisté výhody z pohledu následné implementace.

Systém byl v rámci diplomové práce kompletně implementován a v průběhu implementace konfrontován s připraveným návrhem. Kromě samotné fyzické implementace systému je součástí práce i její podrobný popis, aby bylo možné ji do budoucna dále rozšiřovat. Práce by tak měla sloužit i jako základ pro následné navázání na současnou podobu řešení.

Při implementaci bylo nutné pohybovat se v rámci jistých hranic vymezených jak samotným zadáním, tak omezeními plynoucími z použitých nástrojů a programovacích jazyků. Proto bylo třeba zvažovat využití obecně známých postupů při samotné implementaci a drobně si je upravovat, aby byly použitelné i v rámci zmíněných omezení.

Výsledný systém bude nasazen do ostrého provozu na stránkách projektu FITkit a bude se snažit usnadňovat práci při dalším vývoji celé platformy FITkit tak, jak bylo původně zamýšleno v zadání projektu. Současně bude pomáhat se shromažďováním informací souvisejících s pádem aplikace QDevKit a tím se nepřímo podílet na jejím zlepšování.

I přesto, že systém splňuje všechny požadavky specifikované v zadání, lez předpokládat jeho další rozvoj. A to především rozšířením o nové prvky usnadňující práci s ním, jako je například doplnění existujících vývojových prostředí o moduly pro přímou správu přidělených ticketů, bez nutnosti pracovat s webovým rozhraním. Nebo vytvoření nástroje pro sběr souhrnných informací o ticketech a verzích a jejich následné zobrazení ve formě různých statistik, vhodných při rozhodování o dalším směřování vývoje celého projektu.

Literatura

[1] *RELEASE v tags/trac-0.12.3 – The Trac Project* [online]. Poslední úpravy 13.6.2010 [cit. 1.5.2012].
Dostupný z WWW: <http://trac.edgewall.org/browser/tags/trac-0.12.3/RELEASE>.

[2] *Development Roadmap :: Bugzilla :: bugzilla.org* [online]. Poslední úpravy 25.3.2009 [cit. 1.5.2012]. Dostupný z WWW: <http://www.bugzilla.org/status/roadmap.html>.

[3] *Installation List :: Bugzilla :: bugzilla.org* [online]. Poslední úpravy 10.5.2011 [cit. 1.5.2012]. Dostupný z WWW: <http://www.bugzilla.org/installation-list>.

[4] *Release Information :: Bugzilla :: bugzilla.org* [online]. Poslední úpravy 19.11.2006 [cit. 1.5.2012]. Dostupný z WWW: <http://www.bugzilla.org/releases>.

[5] *About Atlassian - customers, life, community, FedEx days | Atlassian* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <http://www.atlassian.com/company>.

[6] *14,500 Customers and Counting - JIRA* [online]. Poslední úpravy 2012 [cit. 1.5.2012]. Dostupný z WWW: <http://www.atlassian.com/software/jira/customers.jsp>.

[7] *Assembla project workspaces to accelerate software teams, with issue tracking, GIT, SVN and collaboration | Assembla* [online]. Poslední úpravy 2012 [cit. 1.5.2012].
Dostupný z WWW: <http://www.assembla.com>.

[8] *CVS - Open Source Version Control* [online]. Poslední úpravy 3.12.2006 [cit. 1.5.2012]. Dostupný z WWW: <http://cvs.nongnu.org>.

[9] *Apache Subversion Features* [online]. Poslední úpravy 2011 [cit. 1.5.2012].
Dostupný z WWW: <http://subversion.apache.org/features.html>.

[10] *PHP: SVN - Manual* [online]. Poslední úpravy 4.5.2012 [cit. 6.5.2012].
Dostupný z WWW: <http://www.php.net/manual/en/book.svn.php>.

[11] *GettingStartedWithBreakpad - google-breakpad - Getting Started With Breakpad. - Crash reporting - Google Project Hosting* [online]. Poslední úpravy 13.1.2011 [cit. 1.5.2012].
Dostupný z WWW:
<http://code.google.com/p/google-breakpad/wiki/GettingStartedWithBreakpad>.

[12] *Úvod – FITkit* [online]. Poslední úpravy 2012 [cit. 1.5.2012].
Dostupný z WWW: <http://merlin.fit.vutbr.cz/FITkit>.

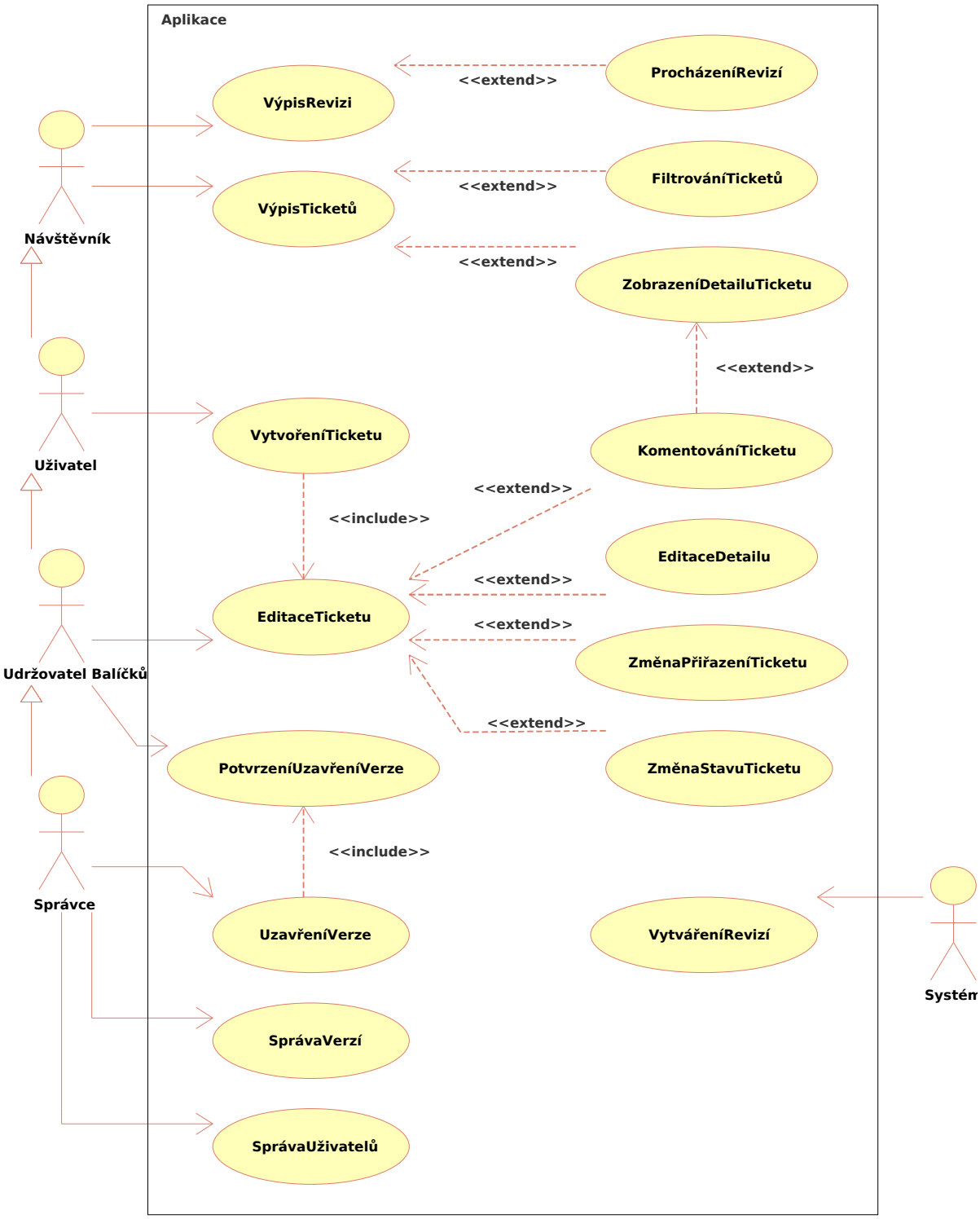
[13] *Static call versus Singleton call in PHP « Hardcoded* [online]. Poslední úpravy 2.3.2010 [cit. 1.5.2012]. Dostupný z WWW:
<http://moisadoru.wordpress.com/2010/03/02/static-call-versus-singleton-call-in-php>.

[14] FOWLER, Martin. Patterns of enterprise application architecture. Boston: Addison-Wesley, c2003, 533 s. ISBN 03-211-2742-0.

[15] *Forms in HTML documents* [online]. Poslední úpravy 24.12.1999 [cit. 1.5.2012].
Dostupný z WWW: <http://www.w3.org/TR/html4/interact/forms.html#h-17.13.4.2>.

Seznam příloh

Příloha A. Diagram případů užití



Příloha B. Případ užití VytvoreníTicketu

ID:	1
Název:	VytvoreníTicketu
Popis:	Uživatel vytvoří nový ticket
Primární aktéři:	Uživatel
Sekundární aktéři:	
Předpoklady:	1 Uživatel je přihlášen
Následné podmínky:	1 Je vytvořen nový ticket
Akce pro spuštění:	Uživatel vybere „Vytvořit nový ticket“
Hlavní tok:	1 Systém zobrazí uživateli formulář na vytvoření nového ticketu 2 Uživatel vyplní požadované údaje 3 Uživatel může připojit k ticketu soubory 4 Uživatel může připojit k ticketu komentář 5 Uživatel zvolí operační systém 6 Uživatel potvrdí vytvoření ticketu 7 Systém vytvoří nový ticket 8 Systém nastaví ticketu status na „nový“
Alternativní toky:	Nevyplněny povinné údaje
Výjimky:	Zrušení

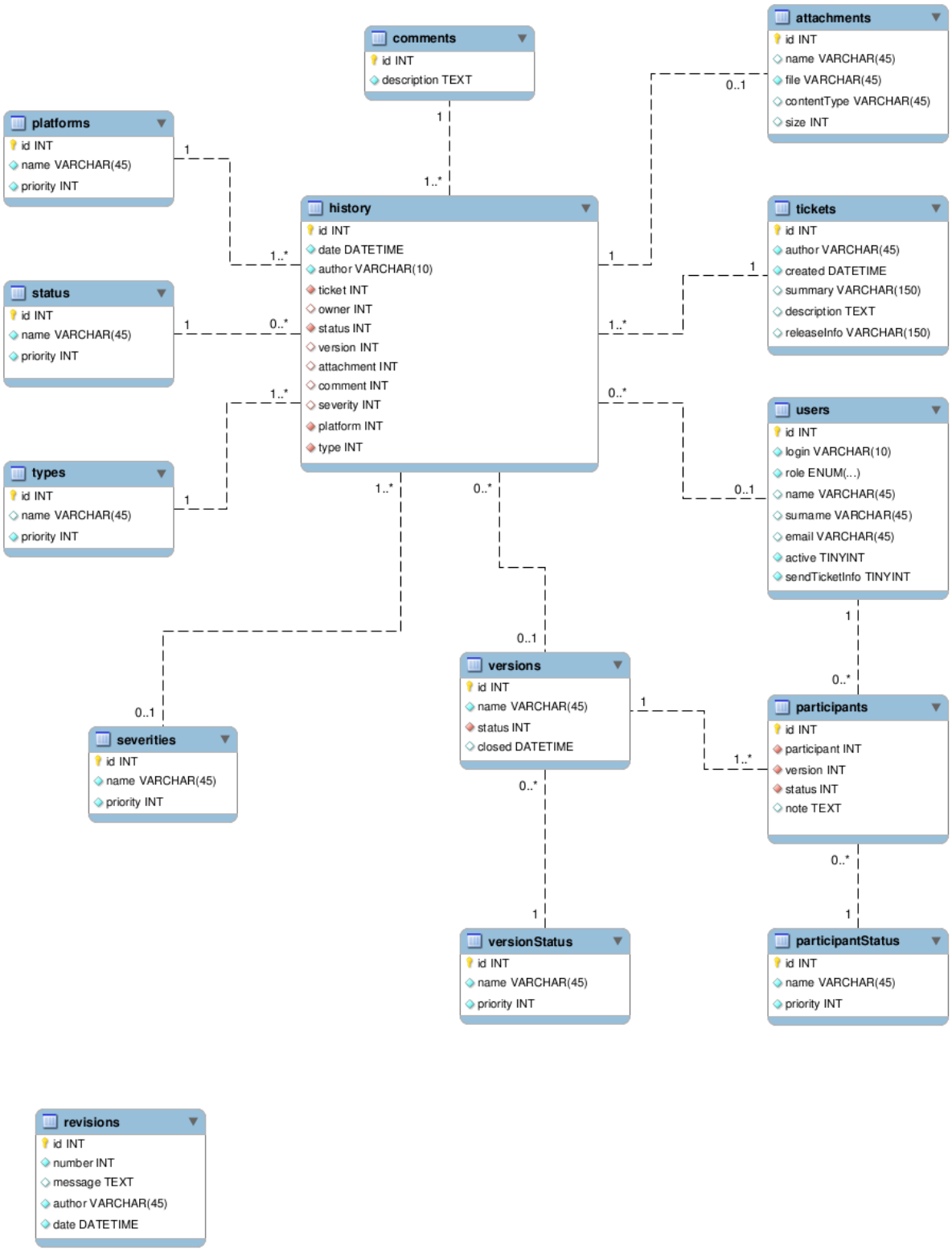
Příloha C. Alternativní tok případu užití VytvoreníTicketu

ID:	1.1
Název:	VytvoreníTicketu: Nevyplněné povinné údaje
Popis:	Uživatel nevyplnil některý z povinných údajů
Primární aktéři:	Uživatel
Sekundární aktéři:	
Předpoklady:	1 Uživatel nevyplnil některý z povinných údajů
Následné podmínky:	1 Je zobrazeno upozornění
Akce pro spuštění:	Uživatel potvrdí vytvoření ticketu v hlavním toku případu 1
Alternativní toky:	1 Systém upozorní uživatele o položkách, které je nutné vyplnit 2 Návrat k bodu 2 hlavního toku
Frekvence:	Občas

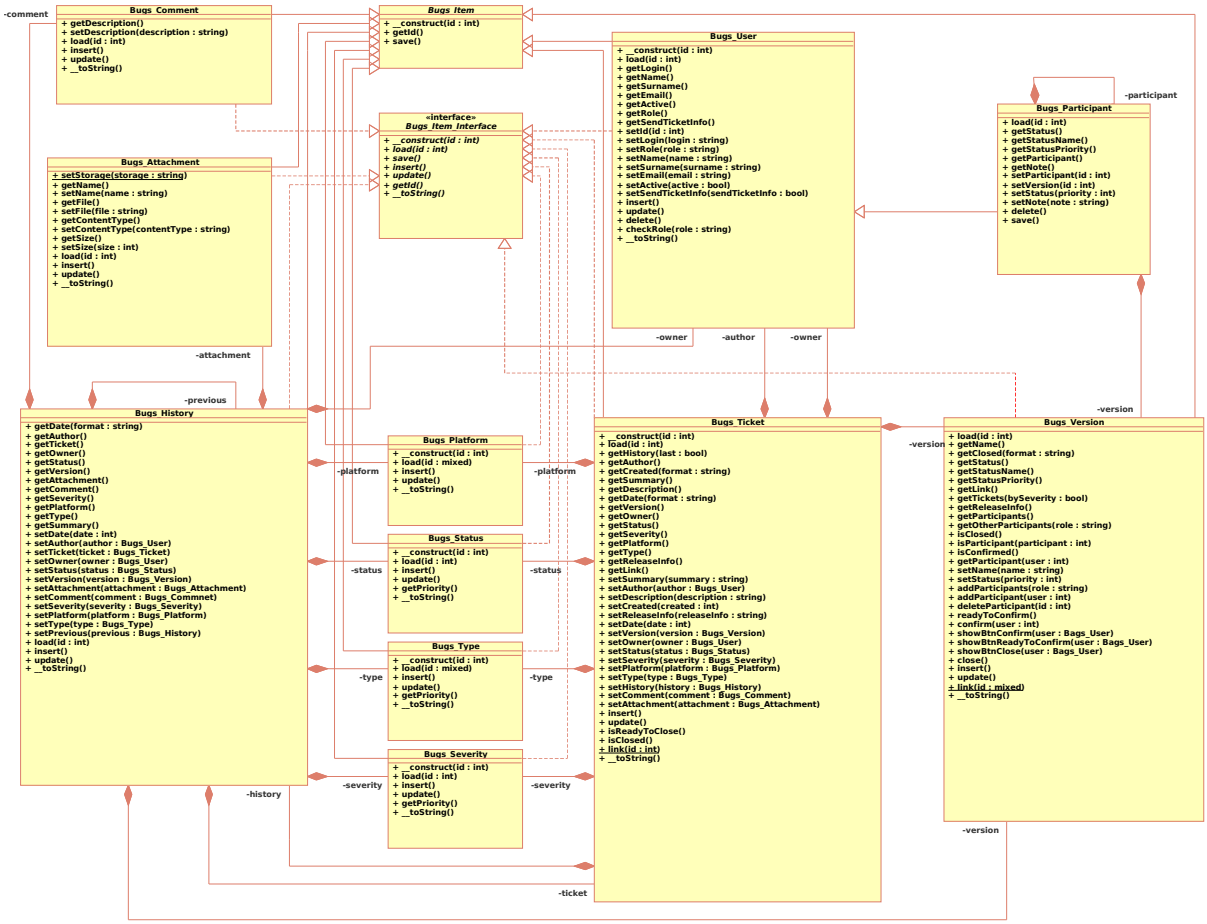
Příloha D. Případ užití UzavřeníVerze

ID:	2
Název:	UzavřeníVerze
Popis:	Uzavření vývojové verze
Primární aktéři:	Správce
Sekundární aktéři:	Udržovatelé balíčků
Předpoklady:	1 Všechny tickety v dané verzi jsou uzavřeny
Následné podmínky:	1 Je uvolněna nová verze
Akce pro spuštění:	Správce označí danou verzi za připravenou k uzavření
Hlavní tok:	1 Systém rozešle e-mail všem udržovatelům balíčků 2 Všichni udržovatelé balíčků označí verzi za připravenou k uzavření 2.1 Udržovatel balíčku přeloží balíček pro svou platformu 2.2 Udržovatel balíčku označí verzi za připravenou k uzavření 3 Systém informuje správce o potvrzení uzavření od všech udržovatelů balíčků 4 Správce označí verzi za uzavřenou 5 Systém nabídne novou verzi v informačním kanálu pro všechny aplikace
Alternativní toky:	
Výjimky:	
Frekvence:	Občas

Příloha E. Databázový návrh

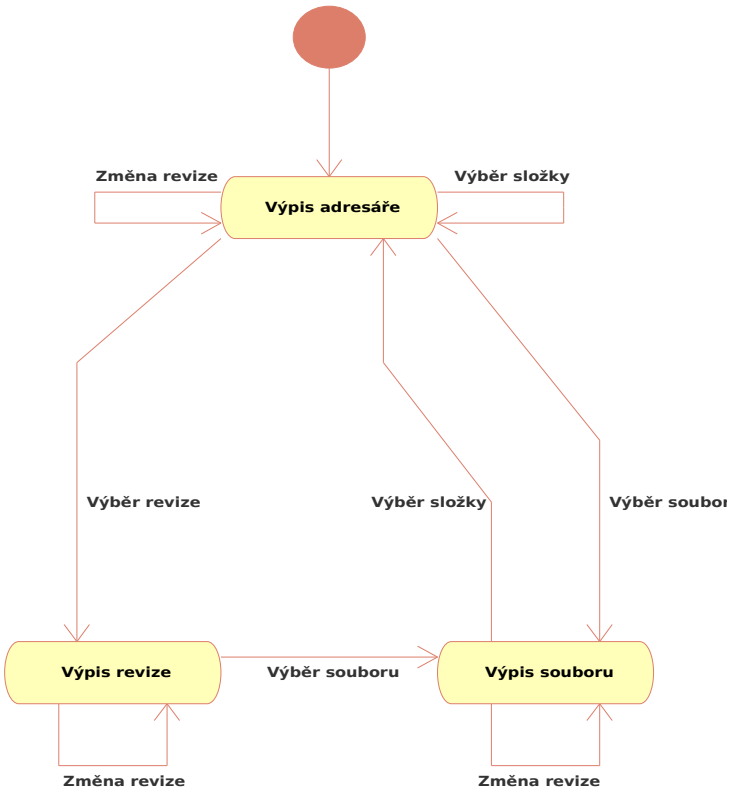


Příloha F. Diagram tříd Bugs_Item



Příloha G. Návaznost obrazovek

Výpis repozitáře



Příloha H. Návaznost obrazovek

Správa ticketů